Universitat de Lleida

# TREBALL FINAL DE GRAU

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Albert Barrufet Bravo

Titulació: Grau en Enginyeria Electrònica Industrial i Automàtica

Títol de Treball Final de Grau: Desenvolupament d'una llibreria de components compatibles amb circuits integrats CMOS de la sèrie 4000 per implementar circuits digitals mitjançant l'ús de FPGAs lliures.

Director/a: **Albert Saiz Vela**

Presentació

Mes: Juliol

Any: 2020

## Agraïments

En primer lloc m'agradaria agrair als meus pares i a la meva germana tot el suport que m'han donat, la formació personal que m'ha ajudat a ser qui sóc i també el suport durant l'etapa acadèmica que amb aquest treball culmina.

També he de citar en aquest aspecte als meus amics, per ajudar-me a desconnectar quan m'ha fet falta i estar sempre al meu costat.

I finalment a la persona més important per realitzar el treball que és el meu tutor Albert Saiz, que ha estat sempre disponible i amb una solució a qualsevol problema que he tingut, per la seva atenció i al mateix temps la seva flexibilitat en un moment realment difícil per tota la societat.

# Acrònims

| Acrònims | |
|---|---|
| FPGA | Field-Programmable Gate Array |
| ASIC | Application-Specific Integrated Circuit |
| PLD | Programmable Logic Device |
| CLB | Configurable Logic Block |
| SOPC | System On a Programmable Chip |
| SDR | Software Defined Radio |
| DSP | Digital Signal Processing |
| OEM | Original Equipment Manufacturer |
| RAM | Random Access Memory |
| HDL | Hardware Description Language |
| ALGOL | Algorithmic Language |
| VHDL | VHSIC Hardware Description Language |
| VHSIC | Very High-Speed Integrated Circuits |
| IEEE | Institute of Electrical and Electronics Engineers |
| RCA | Radio Corporation of America |
| TTL | Transistor-Transistor Logic |
| CMOS | Complementary Metal-Oxide Semiconductor |
| MOSFET | Metal Oxide Semiconductor Field Effect Transistor |
| MSB | Most Significant Bit |
| LSB | Least Significant Bit |
| MEMS | Sistemes microelectromecànics |
| LED | Light-emitting diode |
| PCB | Printed Circuit Board |
| BOM | Bill Of Materials |
| DIP | Dual In-line Package |

# Índex

# Índex de figures

# Índex de taules

# Introducció

La metodologia de l'aprenentatge té moltes variants, i sovint es presenten diferents opcions adequades per una mateixa situació. En l'àmbit de l'enginyeria, un dels punts més importants és saber aplicar els coneixements teòrics adquirits i aconseguir plasmar-los a la realitat mitjançant activitats pràctiques.

Sovint s'utilitzen simuladors que permeten als estudiants fer el disseny inicial del projecte, que en el cas de l'electrònica digital sol ser un circuit amb blocs seqüencials i combinacionals interconnectats.

Els simuladors permeten per una banda fer comprovacions amb una certa facilitat i solen detectar els errors de disseny que puguin tenir els alumnes, per tant és una eina fonamental per traslladar els conceptes teòrics a un disseny funcional.

En l'electrònica digital es sol utilitzar el Software Proteus, que és un dels programes de disseny de circuits electrònics més potents i amb una gran biblioteca pel que fa als components digitals.

El següent pas en l'aprenentatge és traslladar el disseny de l'ordinador a un Hardware on els estudiants troben les dificultats de fer un disseny físic. En l'àmbit universitari actualment s'utilitza la implementació amb "protoboards" i circuits integrats comercials, que és un model d'aprenentatge clàssic però eficaç. De forma relativament recent s'està introduint l'ús de FPGAs, que tenen l'avantatge de suprimir les interconnexions entre blocs, ja que es poden programar sobre el hardware. A més apropa als estudiants al món dels microprocessadors i els microcontroladors.

L'ús de les FPGAs està a l'ordre del dia en l'àmbit professional i tenen la rara característica en el sector de l'electrònica de ser reprogramables, per tant també tenen el valor afegit de la seva versatilitat a l'hora de poder implementar diferents pràctiques en una mateixa placa sense modificar el hardware.

## Objectius

D'acord amb el que s'ha mencionat en la introducció sobre els avantatges de les FPGA els objectius són:

- **Objectius principals:**

  - Creació d'una biblioteca de components que permeti ampliar les opcions de disseny de circuits digitals amb FPGAs utilitzant components de la família CMOS 4000.
  - Treballar amb eines de software i hardware lliure que permetin als estudiants treballar tant des de casa com des de la universitat sense cap cost.

- **Objectius secundaris:**

  - Escollir programes i eines adequades per poder treballar correctament i que siguin útils per realitzar activitats pràctiques.
  - Analitzar diferents llenguatges de descripció de Hardware i veure quin és el més òptim per aconseguir els objectius principals.
  - Fer algun disseny que demostri les facilitats que ofereix un hardware reprogramable enlloc del model clàssic de pràctiques.

# CAPÍTOL I: FPGA

## 1.1 Introducció

Les FPGAs (field-programmable gate array) són matrius de portes lògiques programables en camp que permeten configurar-les per implementar hardware en elles. Els dispositius FPGA són circuits integrats en un dispositiu semiconductor que permeten a l'usuari programar-los tantes vegades com es desitgi per reproduir les funcions que vol el dissenyador o programador.

Les FPGA van aparèixer a mercat per primera vegada el 1985, inventades pels cofundadors de Xilinx, Ross Freeman i Bernard Vonderschmitt. Inicialment van aparèixer com una combinació entre els dispositius lògics programables (PLD) i els circuits integrats d'aplicació específica (ASICs). La primera placa que van llançar al mercat tenia 64 blocs lògics configurables (CLBs), mentre que ara ja en tenen milions.

Actualment, s'utilitzen en aplicacions similars als ASICs (circuits integrats d'aplicació específica) encara que són més lentes, tenen un major consum de potència i no poden contenir sistemes tan complexos com elles mateixes. Així i tot, les FPGAs tenen els avantatges de ser reprogramables (el que afegeix una gran flexibilitat al flux de disseny), els seus costos de desenvolupament i adquisició són molt menors per a petites quantitats de dispositius i el temps de desenvolupament és també menor respecte als ASICs. [1]

## 1.2 Utilitats

Les FPGAs poden ser utilitzades per solucionar qualsevol problema computable. Això va quedar demostrat, ja que les FPGA es poden programar com processadors softcore, també coneguts com a SOPC, que bàsicament consisteixen en implementar un processador en un sistema reprogramable. En són un exemple la Xilinix MicroBlaze o la Atera Nios II. [2]

A més tenen la capacitat de realitzar alguns càlculs de forma simultània (computació paral·lela), el que fa que en siguin significativament més ràpides realitzant determinats processos que un microprocessador. [3]

Una altra tendència en l'ús de les FPGA és l'acceleració de hardware, on s'utilitzen per compartir part de la computació amb un processador genèric agilitzant d'aquesta manera certes parts d'alguns algoritmes. El cercador "Bing", per exemple, és conegut per utilitzar FPGAs per accelerar el seu algoritme de cerca des de 2014. [1]

## 1.3 Aplicacions

El més interessant d'aquests dispositius és la quantitat d'aplicacions que tenen actualment, ja que degut a les seves característiques poden anar enfocats a sectors com:

- **Aeroespacial i defensa:** Aviació/ED-80, comunicacions, míssils i municions, s'utilitza també per aplicacions en l'espai exterior pel fet que poden ser resistents contra la radiació ionitzant.

- **Àudio:** Convertidors Digital-Analògic, reconeixement de la parla, ràdio definida per Software (SDR), processament de senyals digitals (DSP), solucions de connectivitat i electrònica portàtil.

- **Automoció:** Xarxa i connectivitat de vehicles, processament d'imatge (càmeres dels cotxes).

- **Emissions TV:** Classificació de colors, codificadors de pantalla, processament de vídeo en temps real, commutadors i routers.

- **Electrònica:** Pantalles i càmeres digitals, impressores multifuncionals, electrònica portàtil, receptors de televisió i cartutxos amb memòria flash.

- **Centres de dades:** Servidors, Seguretat, Mòdul de seguretat de hardware, routers, interruptors, portes d'enllaç i balanç de càrrega.

- **Computació d'alt rendiment:** Servidors, supercomputadores, mineria de dades, conformació de feixos (beamforming), radars d'alta gamma, i intel·ligència de senyals.

- **Industrial:** Xarxes industrials, processament d'imatge per la indústria i control de motors.

- **Mèdic:** Ultrasons, raigs X, tomografia per emissió de positrons, tomografia computada, cirurgia remota i imatges per ressonància magnètica.

- **Instrumentació científica:** Radioastronomia, Amplificadors Lock-in que permeten obtenir senyals quan es coneix la seva portadora en entorns amb molt soroll i en llaços de seguiment de fase.

- **Comunicacions amb i sense cable:** Xarxes de transport òptic, processament de xarxes, connectivitat d'interfícies, banda base, radio, xarxes de retorn (Backhaul). [1]

Es pot veure la gran quantitat d'aplicacions que té en el camp de l'electrònica, especialment pel que fa al processament de senyals i el tractament d'àudio i d'imatge. Cal tenir en compte que aquesta és una llista dinàmica que es va actualitzant.

## 1.4 Característiques tècniques

Les FPGAs tenen la característica atípica en el sector de ser reprogramables, que els afegeix un atractiu innegable a l'hora d'aconseguir més flexibilitat. Tot i això el mercat de les FGPAs els darrers anys està centrat en centres de dades, la intel·ligència artificial i els avenços per millorar els processadors. En aquests camps la companyia Intel està fent una forta aposta per les FPGAs. [1]

Dins de la flexibilitat que proporcionen les FPGAs cal tenir en compte que hi ha tant la variant de fer-hi canvis físics, com la d'aprofitar el hardware que tenen i modificar el programa o arxiu que les controla per obtenir diferents funcionalitats. Per tant, si algun sistema generés algun problema o requerís alguna modificació important seria més senzill solucionar-lo si es treballa amb una FPGA, ja que es podria reprogramar. En canvi, un ASIC que en la majoria de casos haurà de ser substituït.

Per tant, cal tenir en compte que aquesta flexibilitat permetria reduir costos de forma significativa a una companyia que hagués de realitzar tasques específiques, ja que amb les FPGAs podria crear un programa que els permetés optimitzar l'ús de la seva potencia i per tant estalviar costos també en la realització d'aquestes tasques.[4]

Una altra característica important de les FPGA és l'acceleració en dos escenaris completament diferents:

- Fabricació: Les FPGA són molt senzilles de construir i es venen a punt per ser programades i utilitzades, el qual implica una reducció en els temps de fabricació totals de l'empresa. Pel que fa al disseny, quan l'empresa ha realitzat el disseny inicial un OEM pot fabricar l'equip en grans quantitats i enviar-los a l'empresa, que posteriorment s'encarregarà de fer la venda al detall, per tant s'estalviarà també aquest temps.

- Acceleració en si mateixa: Les FPGA en si mateixes són elements que ajuden a augmentar el rendiment dels sistemes. Col·laborant amb processadors i alliberant-los de part del treball que aquestes haurien de fer mitjançant acceleracions en la càrrega i descàrrega d'informació i que per tant augmenten el rendiment dels sistemes.

Una altra característica a tenir en compte en les FPGA és el seu nivell de complexitat. Amb el temps s'ha aconseguit integrar dins d'una FPGA motors DSP, RAM, processadors i altres sistemes individuals per realitzar una mateixa matriu, i si segueixen aquest creixement podrien rivalitzar en un futur amb qualsevol dels sistemes bàsics pel que fa a complexitat i a potència bruta. [2]

# CAPÍTOL II: PROGRAMACIÓ D'UNA FPGA

## 2.1 Introducció als HDL

El llenguatge de descripció de hardware és un llenguatge de programació que s'utilitza per definir el disseny i l'estructura dels circuits sense necessitat d'utilitzar un diagrama electrònic.

Això, per tant, permet una descripció formal i precisa d'un circuit electrònic i fa possible fer anàlisis i simulacions d'aquest circuit. Els llenguatges de descripció de hardware s'utilitzen sobretot en circuits electrònics digitals.

Els llenguatges de descripció de hardware són relativament semblants al C, Java o ALGOL, ja que consten de descripcions de text amb expressions, declaracions i estructures de control. Presenta la singularitat que en aquests tipus de llenguatges, a diferència d'altres, s'inclou explícitament una noció del temps. [5]

**Avantatges**

- El seu ús s'ha generalitzat tant pel que fa al disseny com a la síntesi del propi llenguatge.
- Permeten una comunicació comprensible entre diferents equips de treball.
- Els principals HDLs estan estandarditzats i són llenguatges oberts i per tant accessibles pels usuaris.
- Les seves descripcions poden tenir diferents nivells d'abstracció.
- El seu funcionament no depèn de la metodologia emprada per fer el disseny.
- Els codis generats són reutilitzables sobre diferents hardwares (o sobre un mateix hardware).
- No són necessàriament dependents de la tecnologia que s'utilitza.

**Inconvenients**

- Tenen una evolució lenta i amb diferències significatives entre les diferents versions que surten.
- La seva portabilitat de vegades es veu imitada a causa del fet que no presenten una semàntica matemàtica formal. [6]

## 2.2 Principals HDL

Els dos HDL més utilitzats en l'actualitat són VHDL i Verilog, tot i que també cal destacar per exemple el System C, que és una ampliació de C++ que s'utilitza per simular circuits digitals. Existeixen altres llenguatges amb un ús menys estès, com per exemple PALASM, Gezel, Lola... [7]

## 2.3 VHDL

El llenguatge VHDL és un HDL desenvolupat inicialment per Texas Instruments i IBM, que van ser contractats pel departament de defensa dels EUA l'any 1983, amb la intenció de crear una eina que permetés documentar en un únic llenguatge el comportament dels ASICs que les companyies incloïen en el seu equipament. Aquest llenguatge es va estandarditzar per primer cop amb la IEEE 1076-1987, tot i que la versió més utilitzada i amb més eines de suport és la primera gran revisió que es va fer amb el nom IEEE 1076-1993.[8]

Al ser un llenguatge estandarditzat no depèn de cap fabricant i per tant els dissenys es poden mantenir i reutilitzar, a més la seva estandardització també te l'avantatge de què el converteix en llenguatge amb un disseny jeràrquic que manté l'ordre i unes normes.

Dins del llenguatge VHDL hi ha diferents formes de dissenyar un mateix circuit i depèn del programador escollir quina és la millor en cada cas.

- **Funcional o de descripció de comportament:** Especifica el comportament de les sortides respecte de les entrades, no es dóna informació de com serà el circuit.

- **Descripció de flux de dades:** Aquesta opció consisteix a escriure assignacions booleanes de forma concurrent i que descriuen com circula la informació.
- **Descripció estructural:** L'última possibilitat és fer una descripció estructural, que és habitual utilitzar en circuits més complexos. Consisteix a escriure el circuit amb instàncies a diferents mòduls, per exemple, portes lògiques. Es forma així un disseny de jerarquia superior al connectar els ports d'aquestes instàncies amb els senyals interns del circuit.

També existeix la possibilitat d'utilitzar aquestes tres estructures d'una forma mixta en cas que el disseny d'un circuit ho requereixi o es consideri més òptim.

Aquesta versatilitat fa que el llenguatge VHDL sigui un dels més utilitzats, ja que permet dissenyar i simular circuits molt complexos. A més, una altra característica és que al ser un llenguatge que permet la descripció d'un sistema concurrent permet executar-se realitzant operacions en paral·lel, a diferència de la majoria de llenguatges de programació (MATLAB, C, C++, R...). [9]

## 2.4 Verilog

Verilog és l'altre gran HDL que existeix en l'actualitat, i ha estat el llenguatge escollit per realitzar els dissenys en aquest treball. Per tant es farà una explicació més àmplia de les seves característiques i components.

### 2.4.1 Introducció

Verilog és un llenguatge que permet dissenyar i descriure qualsevol mena de sistema digital, una característica indispensable en els llenguatges HDL, i que per tant, permet descriure hardware de qualsevol nivell. [10]

En els seus inicis Verilog va néixer basat en el llenguatge de programació C, ja que es volia aconseguir que tingués una acceptació ràpida i resultés familiar pels programadors que havien de fer dissenys. Verilog, tant pel que fa a operadors del llenguatge com a les paraules reservades, és pràcticament idèntic a C. Les grans diferències respecte a C són que Verilog té una noció temporal explícita (com tots els HDLs) i, a més, no incorpora les estructures, punters i funcions recursives.

Els dissenys en Verilog es fan seguint una jerarquia de mòduls que poden contenir instàncies d'altres mòduls. El disseny dels mòduls es fa definint ports d'entrada, sortida i d'entrada-sortida. Internament es comporten com cables o registres.

Els mòduls, mitjançant sentències, defineixen la relació entre els ports i els cables o registres amb la finalitat d'obtenir el comportament desitjat. [11]

### 2.4.2 Història

Verilog va ser un dels primers HDL que va aparèixer i, per tant, dels primers en fer-se popular. Els seus inventors són Prabhu Goel, Phil Moorby, Chi-Lai Huang i Douglas Warmke,que van llançar la primera versió entre 1983 i 1984. Inicialment els drets de la creació de Verilog pertanyien a Automated Integrated Design Systems, posteriorment coneguda com Gateway Design Automation, companyia que va ser adquirida el 1990 per Cadence Design Systems.

En els seus inicis Verilog era un llenguatge merament descriptiu i que permetia fer simulacions. Posteriorment, es va incorporar les funcionalitats per poder fer dissenys en hardware.

A la dècada dels 90 Verilog era un llenguatge poc utilitzat comparat amb VHDL. Per revertir aquesta situació la companyia Cadence Design Systems va decidir que a partir de 1991 fos un llenguatge obert i l'any 1995 l'IEEE el va estandarditzar amb el nom de IEEE Standard 1364-1995.

La segona estandardització de Verilog va ser l'any 2001, amb la IEEE Standard 1364-2001, que bàsicament va reparar els defectes que havien trobat els usuaris en la primera estandardització de 1995. La tercera estandardització va ser Veriog 2005 (IEEE Standard 1364-2005).

El gran canvi del llenguatge Verilog va ser l'any 2009, quan la seva estandardització es va ajuntar amb la de SystemVerilog amb el nom de IEEE Standard 1800-2009. La versió actual estandarditzada de SystemVerilog/Verilog és la IEEE 1800-2017.

SystemVerilog va néixer amb el propòsit de fer front al gran nombre de HDLs que van aparèixer a principis dels 2000. Verilog-2005 era inicialment un subconjunt de SystemVerilog, que era un llenguatge més complet amb noves funcions i possibilitats pel que fa a la verificació i modelització de dissenys. Després de quatre anys com s'ha mencionat es van fusionar les seves estandarditzacions.[11]

### 2.4.3 Nivells d'abstracció

Verilog és un llenguatge capaç de suportar dissenys amb nivells d'abstracció diferents que es poden separar en tres grups

- **Nivell porta:** També conegut com a nivell estructural correspon a una descripció a baix nivell. Es fa utilitzant portes lògiques primitives (AND,OR,NOT...) amb les seves connexions i afegint les propietats temporals que facin falta a cada porta.

- **Nivell de transferència de registre o RTL:** En dissenys d'aquest tipus s'especifiquen les característiques del circuit mitjançant operacions i transferència de dades entre els registres . Verilog, a més, permet decidir a l'usuari en quin instant vol fer cada operació amb les especificacions de temps. Quan s'utilitza el nivell d'abstracció RTL els dissenys són sintetitzables, aquesta propietat fa que aquest sigui el nivell d'abstracció més utilitzat en HDLs.
- **Nivell de comportament:** Aquest disseny es centra més en la descripció del comportament, ja que és independent de l'estructura. En aquest nivell el disseny es realitza mitjançant algorismes en paral·lel. Els algorismes bàsicament són instruccions que es van executant de forma seqüencial.[12]

### 2.4.4 Elements bàsics del llenguatge

#### *Comentaris*

Els comentaris s'utilitzen com a complement al codi. El llenguatge Verilog permet comentar de dues formes diferents:

- **Comentaris d'una sola línia:** Aquests comentaris es fan afegint els caràcters " // " abans del text que es vol comentar, per exemple:

```
// Això és un comentari d'una línia
```

- **Comentaris sense límit de línies:** Aquests comentaris permeten comentar més d'una línia, a l'inici del comentari cal posar els caràcters " /* " i al final del text comentat " */ ", per exemple:

```
/* Això és un comentari
   de diverses línies */
```

#### *Identificadors*

Els identificadors en Verilog tenen la característica que distingeixen entre majúscula i minúscula, i consten únicament de lletres, nombres i els caràcters "_" i "$", els identificadors sempre hauran de contenir una lletra o "_".

```
reg B;
reg A3;
reg A_i;
```

***Nombres***

Verilog permet especificar el valor numèric en 4 bases diferents: binari, octal, decimal i hexadecimal, els nombres negatius es representen en complement a2. La sintaxi de representació d'un nombre és la següent:

```
<MIDA>  <BASE>  <VALOR>
```

La mida és una dada opcional, i que sempre s'expressa en decimal. En cas de no donar-se es suposarà que són 32 bits. La base per defecte i si no s'indica el contrari és la decimal.

```
'b Base binària
'd Base decimal
'h Base hexadecimal
'o Base octal
```

Pel que fa als valors binaris cal destacar que Verilog té 4 valors que pot tenir qualsevol senyal, a més d'un "1" i "0" lògics, també hi ha la possibilitat d'obtenir els valors X (valor qualsevol desconegut) i Z (valor d'alta impedància). El valor Z apareix quan un wire no està connectat.

```
243         // Nombre decimal (Utilitzant 32 bits)
8'h0a       // Nombre o hexadecimal (Emmagatzemat 00001010)
3'b10       // Nombre binari 3 bits (Emmagatzemat 010)
'o73        // Nombre octal (Utilitzant 32 bits)
2'bx1       // Nombre de 2 bits (Emmagatzemat x1)
```

### *Tipus de dades*

Verilog inclou 5 tipus de dades diferents, els més utilitzats son reg i wire.

- **reg:** Representen variables amb capacitat d'emmagatzemar informació
- **wire:** Representen connexions estructurals entre components. No tenen capacitat d'emmagatzematge.
- **integer:** Registre de 32 bits
- **real:** Registre capaç d'emmagatzemar nombres en coma flotant
- **time:** Registre sense signe de 64 bits

Per representar les dades s'utilitza la següent sintaxi:

```
<TIPUS> [<MSB> : <LSB>] <NOM>;
```

Per defecte les variables són d'un sol bit, per tant MSB i LSB només cal especificar-los en cas de definir un vector de bits.

### *Operadors*

Els operadors de Verilog és poden separar depenent del seu tipus:

- **Binaris aritmètics**
  - Suma o signe positiu "+": Pot actuar com a símbol si es situa davant d'una expressió i també per indicar que hi ha una suma entre dos nombres.

```
A = B + C; // B: 4'b1000   C: 4'b01
// Resultat A: 4'b1001
D = +8'h2C; // Resultat D: 8'h2C
```

  - Resta o signe negatiu "-": Pot actuar com a símbol si es situa davant d'una expressió i també per indicar que hi ha una resta entre dos nombres.

```
A = B - C; // B: 4'b1000   C: 4'b10
// Resultat A: 4'b110
D = -8'h01;// Resultat D: 8'hFF (1 en complement a2)
```

  - Multiplicació "*": Multiplica dos nombres de qualsevol tipus.

```
A = B * C; // B: 4'b0100   C: 4'b11
// Resultat A: 4'b1100
```

  - Divisió "/": Divideix dos nombres de qualsevol tipus.

```
A = B / C; // B: 12   C: 3
// Resultat A: 4
```

- Residu "%": Retorna el residu de la divisió de dos nombres de qualsevol tipus.

```
    A = B % C; // B: 4'b1000   C: 4'b10
// Resultat A: 4'b0
```

- **Relacionals**
  - Igualtat "==": Compara dos elements i retorna 1 ó 0, verdader o fals de forma respectiva. Retorna verdader quan els dos elements son iguals.
  - Desigualtat "=!": Funciona de forma inversa a la igualtat, és a dir, retorna verdader quan els dos elements són diferents.
  - Igualtat "===" i desigualtat "==!": Funcionen de forma idèntica a les igualtats ja explicades però també comparen els valors indefinits "X" i d'alta impedància "Z".

```
if (B != A)... // A: 4'b1110   B: 4'b1101
// Resultat: true
if (B === A)... // A: 4'b11X0   B: 4'b11X0
// Resultat: true
```

  - Major i menor "<","<=",">",">=": Serveixen per comparar dos elements i si un respecte de l'altre és més petit, més petit o igual, més gran i més gran o igual respectivament.

```
if (B > A)... // A: 4'b1101   B: 4'b1100
// Resultado: false
```

- **Lògics**
  - Negació "!": Canvia el valor lògic de l'operand al qual precedeix.
  - AND lògica "&&": S'obtindrà el resultat de la combinació entre els operands, en aquest cas obtindrem "true" quan tots dos operands siguin "true" i "false" per la resta de casos
  - OR lògica "||" :Igualment s'obtindrà el resultat de la combinació entre els operands. En aquest cas s'obtindrà "true" sempre que algun dels dos operands ho sigui i "false" si els dos operands també són "false"

```
A: true   B: false
C = (!A)      // Resultat: false
D = (A && B) // Resultat: false
E = (A || B) // Resultat: true
```

- **Lògica de bit**
  - AND "&": Fa la funció AND bit a bit
  - OR "|": OR bit a bit.
  - XOR "^": XOR bit a bit.
  - Negació "~": Negació bit a bit. A més la negació es pot combinar amb les altres 3 portes obtenint NAND (~&), NOR(~|) i NOT XOR (~^).

```
//A:1010 B: 4'b1110
C = A & B; // Resultat C: 4'b1010
D = A | B; // Resultat D: 4'b1110
E = A ^ B; // Resultat E: 4'b0100
F = ~A;    // Resultat F: 4'b0101
G = A ~| B;// Resultat G: 4'b0001
```

- **Lògica de reducció**
  - AND "&": Fa la funció AND de tots els bits de l'operand.
  - OR "|": OR de tots els bits de l'operand.
  - XOR "^": XOR de tots els bits de l'operand.

  La negació (~) es pot combinar amb els altres 3 símbols de lògica de reducció obtenint NAND (~&), NOR(~|) i NOT XOR (~^). En aquest cas la negació en si mateixa no es pot utilitzar per la lògica de reducció.

```
//A:1010
B = &A; // Resultat B: 0
C = |A; // Resultat C: 1
D = ^A; // Resultat D: 1
E = ~&A;// Resultat E: 1
```

- **Altres**
  - Concatenació "{,}":Permet concatenar dos operands
  - Desplaçament "<<" ó ">>": Desplaçament a l'esquerra i a la dreta respectivament. Quan es produeix un desplaçament s'afegeixen zeros per moure els bits en la direcció escollida.
  - Condicional "?": Depenent del valor lògic retornarà un valor o un altre.

```
// A: 4'b110 B: 4'b10
C = {A, B}; // Resultat C: 8'b110_0010
D = A << 2; //Resultat D: 4'b1000
E = B >> 1; //Resultat E: 4'b1
// F: 1'b1 G: 3'b111 H: 3'b101
F == 1? G : H; //Resultat F: 3'b111
```

### 2.4.5 Mòduls

En el llenguatge Verilog s'utilitzen mòduls per posar el codi. No es permet crear variables globals que puguin pertànyer a diferents mòduls. Tot i això, els mòduls poden tenir entrades i sortides que els permeten interconnectar-se.

Els mòduls tenen sempre la mateixa estructura i han de tenir dins un mínim d'elements per tal de poder tenir un correcte funcionament. [13]

```
module <nom> (<senyals>);
<declaració de senyals>
<funcionalitat del mòdul>
endmodule
```

Dins del mòdul es pot distingir entre dos processos:

- **Initial:** És un procés que només s'executa una vegada a l'inici i no té retards, és útil per exemple a l'hora de donar valors inicials a les variables en un testbench tot i que no és sintetitzable.

```
initial
 begin
   ..... // Sentencies
 end
```

- **Always:** És un procés que es va repetint en bucle en funció dels esdeveniments o cada cert temps, com el seu propi nom indica està sempre en execució.

```
always [<temporització> | <@(lista sensible>]
 begin
   ..... // Sentencies
 end
```

Dins dels processos, especialment en els bucles "always" existeixen diferents estructures de control, les més utilitzades són:

- **If / Else:** És una estructura condicional que permet a l'usuari controlar quan vol executar determinades sentencies.

```
// val: 1'b1
  if (val == 1)
    A = 4'b0010;
  else
    A = 4'b1101;  // Resultat A: 4'b0010
```

- **Case:** És una estructura que avalua una expressió i retorna diferents valors en funció dels resultats obtinguts, funciona de forma similar al If-Else però en casos amb moltes possibilitats és més òptim.

```
// val: 2'b01
  case (val)
    2'b00:   A = 4'b0011;
    2'b01:   A = 4'b1100;
    default: A = 4'b1111;
   endcase  // Resultat A: 4'b1100
```

- **For:** L'estructura for permet a l'usuari executar unes sentencies un determinat nombre de vegades o fins que una variable arribi a un determinat valor.

```
  for(i=0; i<64; i=i+1)
  begin
    A[i] = i;
    B[i] = 0;
  end
```

- **While:** L'estructura while realitza de forma infinita les sentències que contingui sempre que es compleixi la seva condició de funcionament.

```
while (val != 4'b1001)
begin
  A = A + 1;
  if (val[0] == 1'b1)
    B = C;
end
```

Les principals estructures funcionen de forma gairebé idèntica al llenguatge de programació C. Cal destacar que hi ha altres estructures que s'utilitzen de forma menys comú:

- **Repeat:** L'estructura repeat té com a condició un número que és la quantitat de vegades que es repetiran les sentències del seu interior.
- **Forever:** Les sentencies dins d'aquesta estructura es repeteixen infinitament, és útil per exemple quan es vol dissenyar un rellotge.
- **Casez i casex:** Funcionen de la mateixa forma que l'estructura case, però en el cas de casez, "Z" es considera una variable indiferent. Casex considera indiferents les variables "Z" i "X"
- **Wait:** Permet interrompre l'execució d'un bucle ja que atura el programa fins que es compleix la condició que posi l'usuari.

### 2.4.6 Jerarquies

Com ja s'ha explicat els dissenys en llenguatge Verilog funcionen mitjançant mòduls. Les jerarquies permeten combinar diferents mòduls per elaborar dissenys complexos a partir d'alguns més senzills. Hi ha dues formes de establir connexions entre mòduls:

- **Per ordre**: Cada senyal es correspon al port d'un mòdul i es realitza de forma implícita.
- **Per assignació:** Es realitza de forma explícita i es relaciona el port amb la variable utilitzant "port.(variable)".

La relació jeràrquica entre els mòduls permet millorar dissenys i augmentar la seva complexitat sense la necessitat de refer-los des del principi, és una eina interessant sobretot tenint en compte que no es poden crear variables globals.

### 2.4.7 Altres característiques

Verilog també té un gran nombre de funcions (amb paraules reservades) que faciliten la programació a l'usuari, com poden ser "display", "fopen", "fclose", "random"...

Una altra característica del llenguatge Verilog és que permet la generació de fitxers "testbench", que, tot i no ser sintetitzables, són molt útils per poder fer verificacions al codi abans d'introduir-lo a un hardware.

També és interessant el fet de que Verilog mitjançant directives permet que un mateix codi tingui comportaments diferents. Les directives més comuns són:

- Include: Permet incloure fitxers que poden tenir altres mòduls i per tant enriquir el comportament del disseny.
- Timescale: Aquesta directiva estableix l'escala de temps amb les que es treballarà. El llenguatge permet anar de 1fs a 100s.
- Define: Permet definir un valor.

# CAPÍTOL III: SOFTWARE I HARDWARE DEL PROJECTE

## 3.1 Software i hardware lliure

La idea de software lliure es refereix a codis oberts que els usuaris poden copiar, modificar i redistribuir el codi font, és a dir, es posa a disposició de l'usuari un codi perquè treballi com consideri convenient. [14]

En aquest projecte s'han utilitzat únicament eines i programari lliure per crear tots els elements, ja que aporta un gran nombre d'avantatges:

- **Col·laboració i revisions del codi:** La comunitat d'usuaris evoluciona el codi i revisen les errades que pugui tenir, d'aquesta forma es té un codi que va evolucionant i mai quedarà estancat degut als interessos d'una empresa o propietari. Això fa que els codis lliures siguin molt fiables perquè la comunitat els va actualitzant i corregint sense dependre del desenvolupador o primer autor.

- **Transparència:** Es pot realitzar un seguiment del codi i de les seves evolucions sense dependre de la informació dels proveïdors, a més de poder conèixer els canvis soferts pel codi.

- **Flexibilitat:** Els codis oberts permeten solucionar una gran varietat de problemes i abordar-los des de diferents punts de vista per obtenir els resultats desitjats. Això no és possible amb codi privat, ja que normalment estan dissenyats per utilitzar-se d'una forma específica i que no permet modificacions directes per part de l'usuari.

- **Menor cost:** Tot i ser codi lliure a vegades cal fer algunes despeses depenent de si es vol reforçar la seguretat, s'utilitza un suport que no és gratuït o alguna altra circumstància. Tot i això el cost sempre serà menor al d'un codi privat que ja inclou totes aquestes despeses en el seu preu inicial.

En aquest projecte no s'ha requerit cap despesa pel que fa al codi ni a cap element de software.

- **Independència dels proveïdors:** No dependre de cap proveïdor dóna la llibertat a l'usuari d'escollir on i quan vol implementar el seu codi. Això fa que un mateix codi pugui ser aprofitat en diferents hardwares obtenint els mateixos resultats. [14]

El hardware lliure per la seva banda fa referència a tots aquells components físics que formen part del projecte. El hardware lliure té unes característiques similars a les del software lliure, ja que ha d'existir la possibilitat d'utilitzar, modificar i redistribuir els components físics sense dependre de cap companyia o particular. Tot i això, la gran diferència és que el hardware s'ha de fabricar i per tant té un cost implícit.

Així doncs, es considera hardware lliure tot component físic del qual es pot disposar almenys dels plànols per fer el disseny. El hardware lliure presenta els següents avantatges respecte del privat.

- **Independència tecnològica:** No es depèn de cap empresa o particular per la fabricació i adquisició del hardware, tot i que sovint és més barat comprar-lo a la companyia que l'ha dissenyat perquè té un cost menor, ja que els fabrica en grans quantitats.
- **Afavoreix la qualitat:** La disponibilitat de plànols i materials així com la interacció entre usuaris són factors que afavoreixen que els components físics estiguin sovint molt ben dissenyats i siguin funcionals en diferents aspectes. En l'àmbit privat solen ser per realitzar funcions molt específiques i els elements de hardware estan pensats per facilitar la fabricació en massa i reduir costos a l'empresa.
- **Reutilització de dissenys:** La reutilització de dissenys (o d'algunes parts) estalvia molt el temps d'estudi i de realització dels elements de hardware, a més permet adaptar-los fent petites modificacions o aprofitar un mateix disseny de hardware per realitzar moltes operacions. En són un exemple els Arduino, que permeten molta varietat de treball incorporant sensors externs i modificant el seu software. [15]

En aquest treball s'ha utilitzat un hardware que es compon d'una placa FPGA lliure adquirida a través de l'empresa que ha fet el disseny i d'una placa d'entrenament dissenyada per la Universitat de Lleida, els dissenys de la qual són accessibles i es poden considerar hardware lliure.

No s'ha de relacionar el concepte de software i hardware lliure amb el concepte de gratuït, com ja s'ha exposat normalment porten un cost associat (el hardware sempre), tot i que segueixen tenint molts avantatges. [15]

Per altra banda treballar amb eines lliures també té alguns inconvenients.

- **Suport al client:** El problema més evident del programari lliure és la inexistència de responsables a qui reclamar o sol·licitar una correcció davant d'un funcionament no esperat.
- **Ús minoritari:** Les empreses i entitats educatives utilitzen majoritàriament software i hardware privat i per tant no sempre és fàcil adaptar les eines lliures a un entorn que majoritàriament fa ús d'eines privades.
- **Inestabilitat:** Sovint les primeres vegades que es crea o es treballa amb software o hardware lliure no ha existit un control de qualitat previ o un estudi en profunditat, el qual provoca que la comunitat d'usuaris vagi trobant els errors i corregint-los amb el pas del temps.
  Tot i això, res assegura que una determinada eina lliure tingui un èxit suficient per mantenir-se activa i millorar amb el temps. En canvi, en el cas de les empreses privades hi ha un control de qualitat previ al llançament dels seus productes i també es garanteix una certa estabilitat. [16]

Tot i els desavantatges, en l'àmbit educatiu el software lliure permet també obtenir als estudiants la satisfacció o motivació de saber que el seu treball pot millorar o complementar els recursos dels quals es disposa, a més ofereix la possibilitat de poder realitzar de forma relativament senzilla projectes de col·laboració amb altres universitats o entitats privades.

## 3.2 Projecte IceStorm

L'enginyera austríaca Claire Wolf (anteriorment coneguda com a Clifford Wolf) va dissenyar mitjançant enginyeria inversa eines de software per tal de poder programar els models FPGA iCE40 LP/HX 1K/4K/8K del fabricant Lattice Semiconductor. Amb això va néixer el projecte IceStorm.

A partir d'aquest procés la comunitat d'usuaris ha anat creant llibreries de components, i eines de software i hardware lliure que han fet créixer el projecte i convertir-lo en una eina molt interessant i relativament potent per treballar amb FPGA i en concret amb electrònica digital.

La *toolchain* del projecte IceStorm permet mitjançant una seqüència d'ordres i programes sintetitzar, implementar (*place and route*) y gravar fitxers bitstream a la FPGA, tal com es pot observar a la Figura 1. La síntesi es realitza mitjançant el programa *Yosis*, que és l'encarregat de transformar el codi de descripció de hardware en llenguatge Verilog (*.v) a una *netlist* en format BLIF (Berkeley Logic Interchange Format) (*.blif). La implementació (*place and route*) es realitza amb el programa *nextpnr* que accepta com a entrada fitxers en format BLIF i genera un fitxer en formato ASCII (*.asc) que conté blocs de 0 i 1 pels bits de configuració de cada cel·la de la FPGA. Finalment, el programa *icepack* transforma el fitxer en format ASCII, generat pel programa *nextpnr* en un fitxer binari (*.bin) i el programa *iceprog* permet gravar la informació binària a la FPGA.[17]

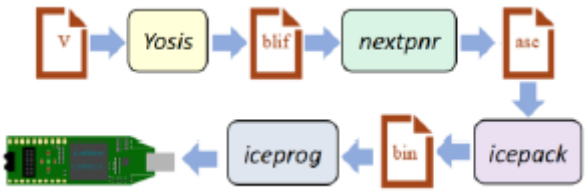


*Figura 1. Toolchain del projecte IceStorm*

Actualment el programa *Yosis* només permet sintetitzar codi utilitzant Verilog tot i que els enginyers Tristan Gingold y Pepijn De Vos estan treballant per poder sintetitzar el codi també en llenguatge VHDL. Com encara no està del tot desenvolupat, la millor forma de treballar amb el projecte IceStorm és utilitzant el llenguatge Verilog. [17]

## 3.3 IceStudio

El programa IceStudio és un editor visual per a FPGAs lliures. Amb el programa IceStudio es poden fer dissenys i sistemes digitals que engloben des d'una simple porta lògica a configuracions més complexes creades a partir de dispositius seqüencials, combinacionals i portes lògiques. [18]

A més, el programa IceStudio permet incloure blocs amb codi Verilog encastat, on es pot descriure el comportament que té el bloc, sent aquest similar al d'un dispositiu comercial (o no). També es pot combinar l'ús de blocs de codi i de portes o altres dispositius. El mateix programa mitjançant la toolchain del projecte IceStorm permet a l'usuari verificar el codi i posteriorment carregar tot el disseny a la FPGA.

Una altra característica important del programa IceStudio és que pot treballar amb diversos models de FPGA. [17]

Actualment les FPGA Lattice iCE40 són les úniques que tenen una *toolchain* totalment lliure i per tant són ideals per poder tenir una gran expansió, fins i tot per davant de les FPGAs de Xilinx o Altera que són els principals fabricants. [19]

Les FPGA Lattice iCE44 compatibles amb IceStudio són:

- HX1K:
  - IceZUM Alhambra
  - Nandland Go board
  - iCEstick Evaluation Kit
- HX8K:
  - IceZUM Alhambra II
  - BlackIce
  - BlackIce II
  - IcoBOARD 1.0
  - Kéfir iCE40-HX4K
  - iCE40-HX8K Breakout Board

- LP8K:
  - TinyFPGA B2
  - Tiny FPGA BX
- UP5K:
  - iCEBreaker
  - iCEBreaker bitsy
  - UPDuino v1.0
  - UPDuino v2.0
  - FPGA 101 Workshop Badge Board
  - iCE40 UltraPlus Breakout Board [20]

Per fer el disseny de les llibreries amb Verilog s'ha escollit la placa IceZum Alhambra II, que utilitza el model HX8K, que és dels més potents i per tant permet fer dissenys més complexos i també té més funcionalitats que, per exemple, la seva predecessora IceZUM Alhambra.

## 3.4 IceZUM Alhambra II

La placa IceZUM Alhambra II és una placa fabricada per l'empresa espanyola Alhambrabits. Aquesta placa té un cost de 60 € i té la important característica de què el seu software i hardware són lliures i és compatible amb el projecte IceStorm. [17][20]


*Figura 2. IceZUM Alhambra II*

Les característiques tècniques de l'IceZUM Alhambra II són les següents:

- iCE40HX4K de Lattice Semiconductors, tot i que realment és un 8K limitat pel software del fabricant, i per tant utilitzant eines lliures es pot aprofitar al màxim passant de 3520 (iCE40HX4K) a 7680 cel·les lliures.
- FTDI 2232 USB que permet programar la FPGA i una interfície UART a un ordinador.
- Interruptor ON/OFF electrònic i que per tant permet encendre i apagar la placa des d'un dispositiu.
- 32MB de memòria flash per un màxim de 30 bitstreams o dades d'usuari
- 20 pins I/O de 3.3 Volts, tot i que també poden tolerar 5V. Cadascun dels pins permet activar de forma directa els LEDs (8 en total), ja que tenen una resistència de 200 Ω en paral·lel.
- Oscil·lador MEMS de 12 MHz.
- Convertidor A/D de 12 bits (4 canals).
- Els pins I/O i d'alimentació tenen protecció permanent contra curtcircuits.

- Botó de reinici estil polsador.
- Alimentació mitjançant dos connectors USB que permeten fins a 4.8V [21]

La placa IceZUM Alhambra II té un pinout similar al d'Arduino UNO amb el propòsit de poder aprofitar algunes plataformes de desenvolupament ja dissenyades per Arduino UNO però en aquest cas controlades a través d'IceZUM Alhambra II. [22]

Com és una eina lliure, els arxius de composició del hardware també estan disponibles i per tant l'usuari té accés als arxius de disseny, que són els esquemàtics, PCB, BOM i arxius GERBER que s'han utilitzat en la fabricació de l'IceZUM Alhambra II.

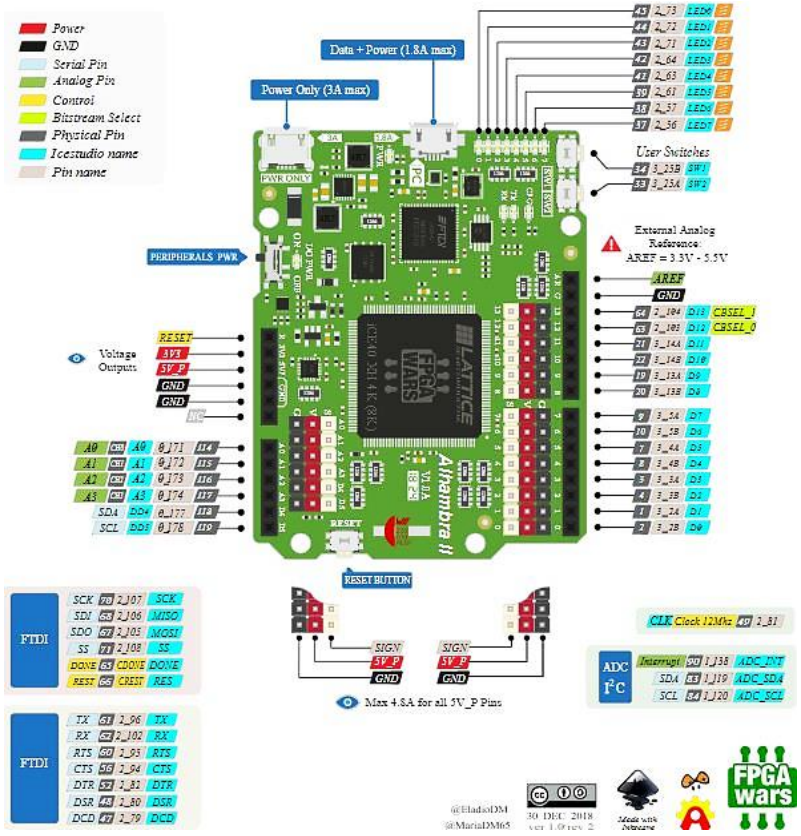


*Figura 3. Distribució de pins i factor de forma de la placa Icezum Alhambra II*

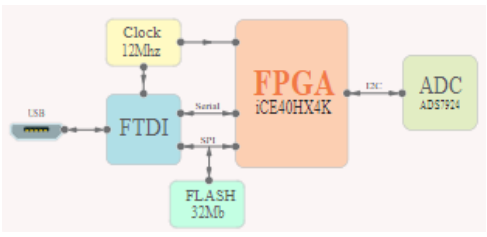


*Figura 4. Connexions i elements d'IceZUM Alhambra II*

## 3.5 Placa d'entrenament

Per tal d'aprofitar al màxim les funcionalitats de la FPGA, la Universitat de Lleida va fer el disseny d'una placa que es pot reprogramar amb l'IceZUM Alhambra II i que té més funcions i possibilitats per realitzar pràctiques que l'IceZUM Alhambra II en si mateixa.

La placa IceZUM Alhambra II disposa de 8 LED's i 2 polsadors. En conseqüència, no hi ha suficients components per poder realitzar pràctiques amb una complexitat adequada al nivell de coneixements que s'ha d'assolir. [17]

La placa d'entrenament com es veu a la Figura 6 consta de:

- 6 LED's de color groc, verd i vermell (2 de cada).
- 4 polsadors.
- 2 interruptors DIP de 4 canals.
- Display de 7 segments controlat per 8 bits independents (7 pel dígit i un punt) .
- 4 displays de 7 segments multiplexats.
- 20 pins I/O
- Alimentació a 3,3V a 5V i GND.

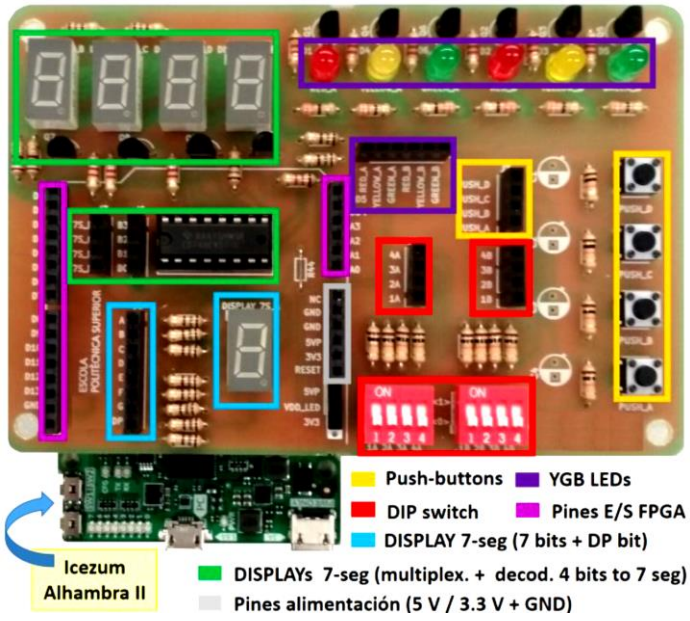


*Figura 5. Disposició dels elements de la plataforma de desenvolupament*

La placa d'entrenament és programable a partir de la FPGA connectant-la a través de les seves connexions femella, ja que realment és una plataforma de desenvolupament de l'IceZUM Alhambra II. La placa presenta les següents característiques tècniques:

- La placa de circuit imprès té unes dimensions de 120 mm x 80 mm el que la fa compatible amb plaques de baquelita i fibra de vidre que tenen un baix cost al mercat.
- Ha estat dissenyada amb el software KiCAD que és lliure i, per tant, permet l'accés a la documentació de la placa i modificar-la o fabricar-la si es considera oportú.
- Per fer les rutes dels components s'ha utilitzat una única capa i per tant és relativament senzilla de fabricar per un particular. En cas de recórrer a una empresa això també reduiria una mica els costos de fabricació. A més els components estan muntats amb el sistema de forat passant, el qual facilita el muntatge als estudiants en un laboratori o a altres particulars.[17]

## 3.6 EDA Playground

EDA Playground és un domini web que ofereix la possibilitat de dur a terme simulacions en diferents HDL d'una forma directa i sense fer ús de cap programa, només fa falta disposar d'un navegador amb connexió a Internet. Ells mateixos defineixen aquest domini web com un espai on accelerar l'aprenentatge del desenvolupament d'un banc de proves amb un accés senzill a les biblioteques i eines EDA. [23]

El simulador EDA Playrgound permet simular SystemVerilog, Verilog, VHDL, C++/SystemC i alguns altres HDL. [23]

En aquest cas per tal de poder simular els diferents blocs a EDA Playground cal generar dos tipus de documents:

- **Testbench:** Aquest fitxer el genera automàticament el programa IceStudio. Bàsicament, permet canviar els valors de les variables al llarg del temps, és un dels avantatges que tenen els HDL, ja que com inclouen una noció del temps explícita faciliten la realització de tests. També cal indicar la freqüència dels rellotges en cas que s'utilitzin en la simulació.

- **Verilog:** Aquest fitxer també és generat automàticament des de l'IceStudio. En aquest cas el fitxer declara les entrades i sortides que tindrà el codi. Hi ha una reproducció del codi que s'escriu dins del bloc que s'ha creat des del programa IceStudio. La principal part d'aquest fitxer és el codi Verilog que descriu el comportament del bloc programat.

El programa IceStudio no genera directament el codi que permet la simulació amb EDA Playground, sinó que genera un fitxer amb l'extensió correcta però que ha de ser lleugerament modificat per tal que el programa funcioni correctament. El resultat final d'aquests codis es pot veure a l'Annex.

Un cop modificats correctament aquests dos fitxers és necessari escollir amb quin llenguatge i simulador es desitja treballar. En el cas dels fitxers creats amb IceStudio cal escollir SytemVerilog/Verilog. Un cop seleccionat el llenguatge cal triar el simulador. Els que funcionen correctament amb Verilog són els simuladors Icarus Verilog. EDA Playground permet escollir entre les versions Icarus Verilog 0.9.6, Icarus Verilog 0.9.7 i Icarus Verilog 0.10.0.

Qualsevol de les 3 versions funciona d'una forma correcta. A més EDA Playground dona l'opció de mostrar EPWave un cop finalitzada la simulació, que és una finestra on es poden seleccionar els senyals que es vulguin veure (tant entrades com sortides).

El resultat que s'obté és una simulació que permet visualitzar els senyals de les variables de sortida en funció de les d'entrada, i mostrar així com funciona el codi i si realment és tal com es desitja. En aquest cas com es simula la família CMOS 4000 la simulació es pot comparar amb el comportament esperat de cada component per assegurar que funciona consultant els seus datasheets.

# CAPÍTOL IV: CARACTERÍSTIQUES DELS CIRCUITS INTEGRATS DISSENYATS

## 4.1 Tecnologia CMOS

Actualment s'utilitzen dues tecnologies d'integració pel disseny de circuits integrats, les dues fan ús essencialment de transistors.

- **Tecnologia TTL:** Lògica de transistor a transistor, aquesta tecnologia utilitza resistències, díodes i transistors bipolars per fer el disseny de les funcions lògiques estàndard.

- **Tecnologia CMOS:** Lògica MOS complementaria, aquesta tecnologia fa ús de l'efecte de camp dels transistors nMOS i pMOS.

**Tecnologia CMOS**

**Avantatges**

- Baix consum de potència estàtica a causa de la naturalesa dels transistors MOSFET, que tenen una alta impedància d'entrada. En estat de repòs només hi haurà corrents paràsits, ja que en cap dels seus estats existeix una connexió directa entre la font de voltatge i el terra.
- Tenen una gran capacitat per aïllar-se del soroll o la degradació del senyal a conseqüència de la impedància metàl·lica de la interconnexió.
- Després de diverses millores, entre les quals cal destacar sobretot la que es va produir entre finals dels anys 70 i inici dels 80, on es van incloure circuits amb noves funcions integrades així com algunes millores en els ja existents, la tecnologia està suficientment desenvolupada per aconseguir grans densitats d'integració i a un preu baix comparat amb altres tecnologies.
- Els circuits amb tecnologia CMOS són relativament fàcils de dissenyar.

**Inconvenients**

- Com a conseqüència de la naturalesa dels MOSFET, que són de caràcter capacitiu la velocitat dels circuits CMOS no és tan elevada com la d'altres famílies lògiques.
- Són vulnerables al latch-up, tot i que amb un bon disseny del circuit el risc de latch-up és pràcticament nul.

  L'efecte latch up és la creació inadvertida d'una resistència elèctrica entre el subministrament de voltatge del MOSFET, el qual crea una estructura paràsit que impedeix el correcte funcionament del circuit que fins i tot pot patir danys en cas de sobrecàrrega.

  En cas de latch-up, si el circuit no ha patit danys es pot reiniciar el sistema per tal que torni a funcionar de forma correcta.
- Quan es redueix molt la mida dels transistors els corrents paràsits comencen a ser comparables als dinàmics, el qual produeix un augment de la potència consumida quan el circuit està en repòs.

## 4.2 Sèrie 4000

La sèrie 4000 és una família de la indústria dels circuits integrats per tal d'implementar funcions lògiques que va aparèixer per primer cop el 1968, introduïda per RCA. També es coneix com a CMOS i és una família que va ser molt utilitzada en els seus inicis. Encara ho és en l'actualitat gràcies al fet que presenta alguns avantatges respecte als circuits integrats TTL de la sèrie 7400.

La sèrie 4000 va néixer com una alternativa versàtil i de baixa potència davant la sèrie 7400 amb tecnologia TTL, totes dues sèries han anat evolucionant però les principals diferències es segueixen mantenint, ja que vénen condicionades per la fabricació. [24]

**Característiques sèrie 4000**

- **Voltatge d'alimentació:** La tensió d'alimentació varia entre 3V i 18V, la sèrie 7400 també té els mateixos límits pel que fa als voltatges d'alimentació, tot i que hi ha subfamílies com la 74HC i 74RCT que utilitzen marges entre 2V i 6V, per això quan s'utilitzen en un mateix circuit integrat la sèrie 4000 i la 7400 amb les seves subfamílies normalment la tensió d'alimentació és de 5 V, aconseguint així que $V_{CC}$ pels TTL y $V_{DD}$ pels CMOS tingui un valor de 5 V.

- **Nivells de voltatge:** Si tant les entrades com les sortides funcionen únicament amb tecnologia CMOS és possible aconseguir uns voltatges de sortida molt propers a 0V, mentre que pel que fa a la tensió d'entrada serà propera al valor de $V_{DD}$.
  Per tant per una tensió d'entrada de 5V els nivells de tensió serien:

| | | |
|---|---|---|
| $V_{OL}$ màxima | 0,05 V | ≈ 0 |
| $V_{OH}$ mínima | 4,95 V | ≈ $V_{DD}$ |
| $V_{IL}$ màxima | 1,5 V | 30% de $V_{DD}$ |
| $V_{IH}$ mínima | 3,5 V | 70% de $V_{DD}$ |

  Amb aquests valors quan el CMOS està en funcionament considerarà baix qualsevol voltatge per sota de 1,5 V i alt qualsevol per sobre de 3,5 V.

- **Immunitat al soroll:** Els circuits amb tecnologia CMOS tenen immunitat al soroll davant d'impulsos que arriben al 30% de la tensió d'alimentació.

- **Rang de temperatures:** La sèrie 4000 pot operar de forma correcta amb unes temperatures entre – 40 ºC i 85 ºC.

- **Temps de propagació:** Els temps de programació varien de forma inversament proporcionat a la tensió d'alimentació, així doncs com més baixa sigui la tensió major serà el temps de propagació.

- **Baix consum de potència:** La característica més important i que marca la diferència respecte d'altres famílies és el baix consum de potència que ofereix la sèrie CMOS 4000. La potència consumida és extremadament baixa sobretot quan el circuit lògic CMOS està en repòs o en estàtic (sense canviar el seu estat). Tot i que la potència consumida va augmentant a mesura que s'incrementa la velocitat de commutació. A continuació hi ha una taula del comportament de diferents famílies per dissenyar circuits d'integració:

| Paràmetre | TTL estàndard | TTL 74L | TTL 74LS | CMOS 4000 $V_{DD}$ = 5 V | CMOS 4000 $V_{DD}$ = 10 V |
|---|---|---|---|---|---|
| Temps de propagació | 10ns | 33ns | 5ns | 125ns | 85 ns |
| Freqüència màxima | 35MHz | 3MHz | 45MHz | 8MHz | 16MHz |
| Potència dissipada | 10mW | 1mW | 2mW | 10nW | 10nW |
| Soroll admissible | 1 V | 1 V | 0.8 V | 2 V | 4 V |
| Fan out* | 10 | 10 | 20 | 50 | 50 |

***El fan out en aquesta taula és el màxim de connexions d'entrada que pot proporcionar una porta lògica de sortida. A vegades el temps de propagació pot condicionar que no s'arribi a màxim teòric.

Amb aquestes característiques i sobretot degut a la seva versatilitat i baix consum de potència, la sèrie 4000 és ideal per utilitzar en sistemes que s'alimenten amb una bateria. Són components ideals sempre que no es requereixi una alta velocitat en el sistema, en cas de prioritzar la velocitat al consum de potència és millor utilitzar la sèrie 7400. [25]

## 4002

- **Entrades:** A1, B1, C1, D1, A2, B2, C2, D2
- **Sortides:** Q1, Q2

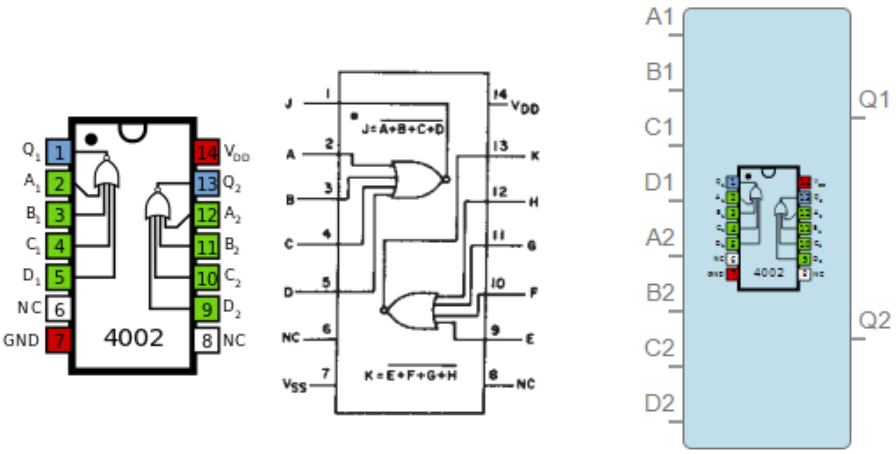


*Figura 6. CMOS 4002 comercial i CMOS 4002 dissenyat amb IceStudio*

En el cas d'aquest bloc totes les entrades fan una funció idèntica. A1, B1, C1 i D1 són les entrades d'una porta NOR que té Q1 com a sortida. La sortida Q2 també és d'una porta NOR que en aquest cas té les entrades A2, B2, C2, D2.

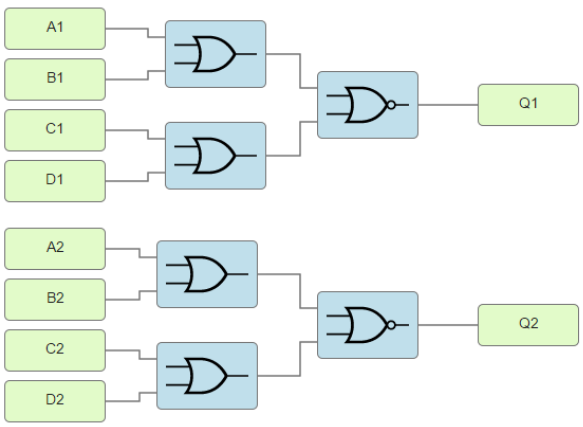


*Figura 7. Disseny del bloc 4002 a IceStudio*

En aquest cas com el programa IceStudio ja incorpora les portes lògiques OR i NOR de dues entrades és suficient amb establir entrades i sortides per reproduir el CMOS 4002.

El bloc 4002 és per tant un conjunt de dues portes NOR de 4 entrades, les portes NOR tenen la següent taula de veritat:

| A | B | C | D | Q |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

*Taula 1. Taula de la veritat porta NOR de 4 entrades*

## 4011

- **Entrades:** A1, B1, A2, B2, A3, B3, A4, B4
- **Sortides:** Q1, Q2, Q3, Q4

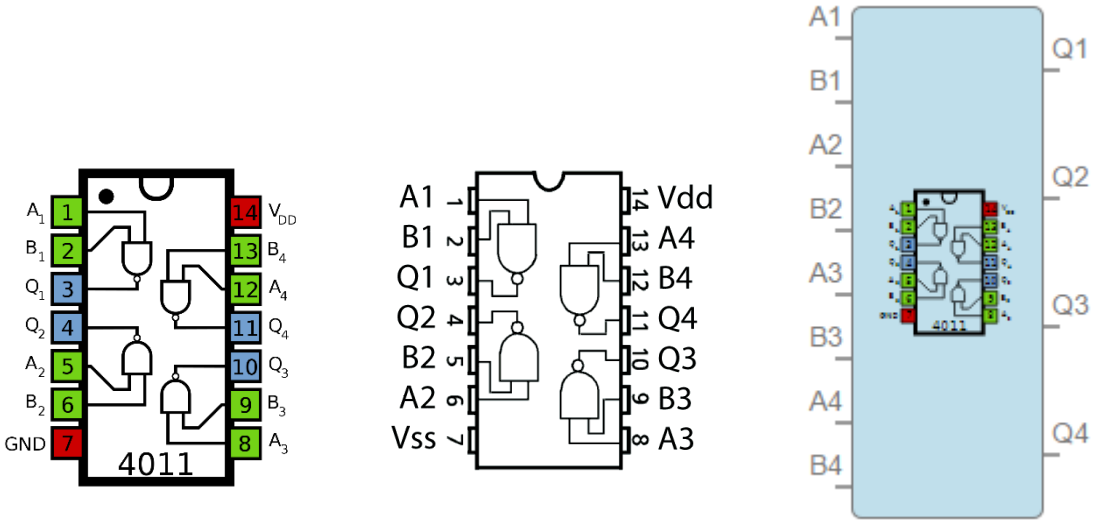


*Figura 8. Bloc CMOS 4011 comercial i dissenyat amb IceStudio*

Les entrades del 4011 són entrades de portes NAND, en aquest cas les entrades amb el mateix número, per exemple A3 i B3 són entrades de la mateixa porta. Les sortides són el resultat (1 o 0 lògic) de la combinació de les dues entrades que tinguin el mateix subíndex. La taula de la veritat d'una porta NAND de dues entrades és la següent:

| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*Taula 2. Taula de la veritat porta NAND de dues entrades*

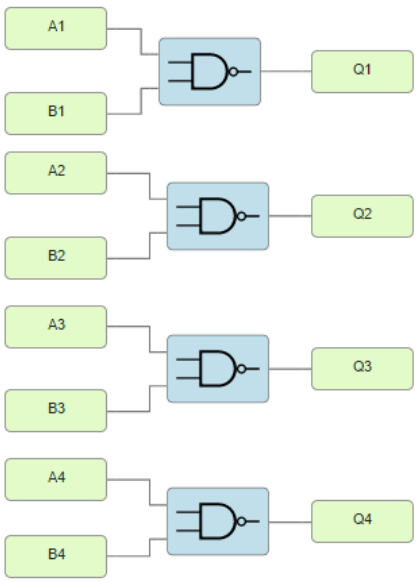


*Figura 9. Disseny intern del bloc 4011 amb IceStudio*

## 4013

- **Entrades:** D1, SET1, RESET1, CLK1, D2, SET2, RESET2, CLK2
- **Sortides:** Q1, NQ1, Q2, NQ2

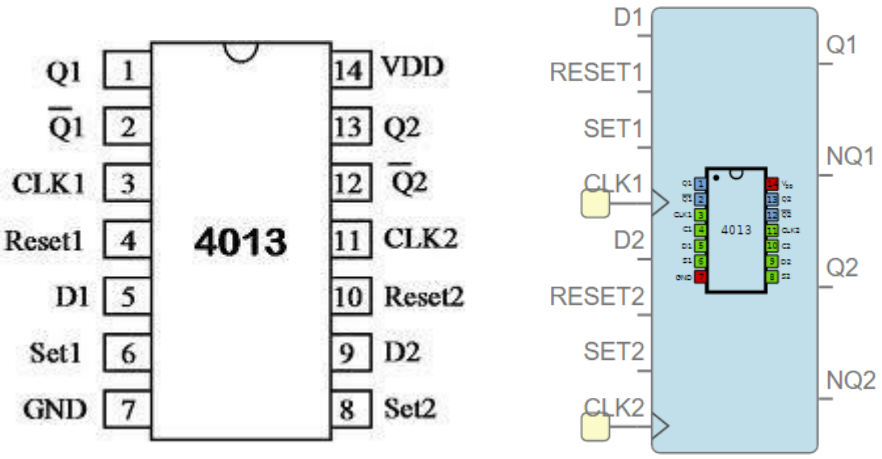


*Figura 10. Bloc comercial 4013 i disseny amb IceStudio*

El bloc 4013 conté dos flip-flop D amb set i reset en el seu interior, per tant les entrades són les d'un flip-flop D amb set i reset però estan repetides. Així doncs, les entrades D1, RESET1, SET 1 I CLK 1 pertanyen al primer flip-flop, que en aquest cas té com a sortides Q1 i la seva negada NQ1. El segon flip-flop funciona de forma idèntica.

A l'hora de fer l'estudi d'aquest bloc és suficient estudiant el comportament d'un dels dos flip-flops, ja que són totalment independents entre ells i l'estat del flip-flop 1 no condiciona en res el del flip-flop 2 i viceversa.

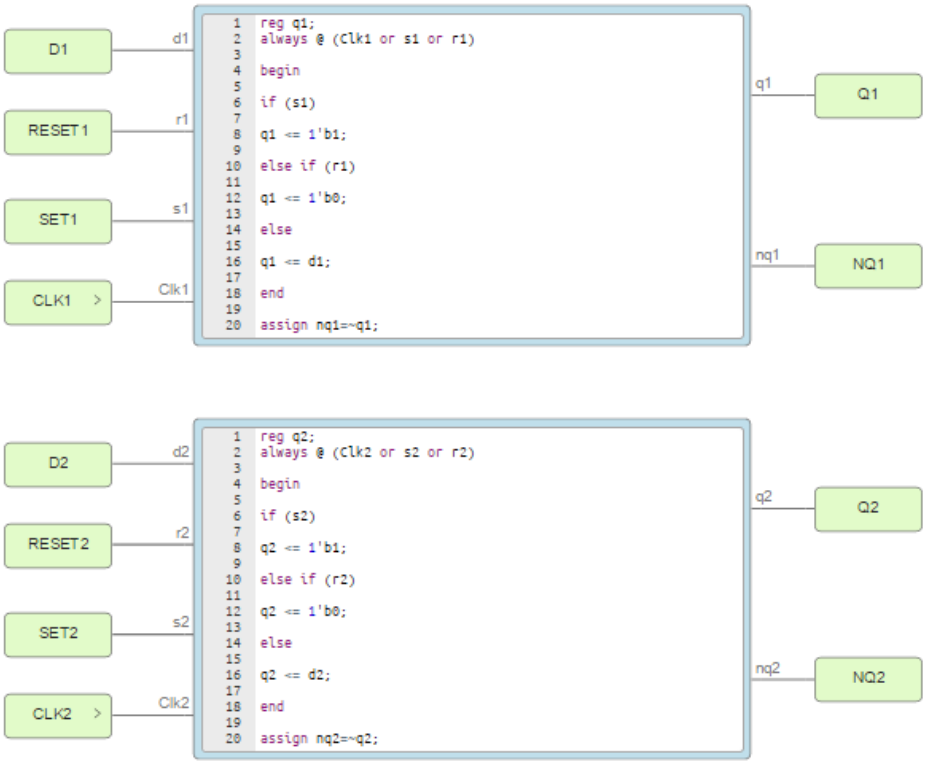


*Figura 11. Disseny intern del bloc 4013*

A la Figura 12 es pot observar el comportament que té un Flip-Flop amb Set i Reset, el rellotge canvia d'estat cada quatre segons i es pot observar que tant el Set com el Reset funcionen correctament.

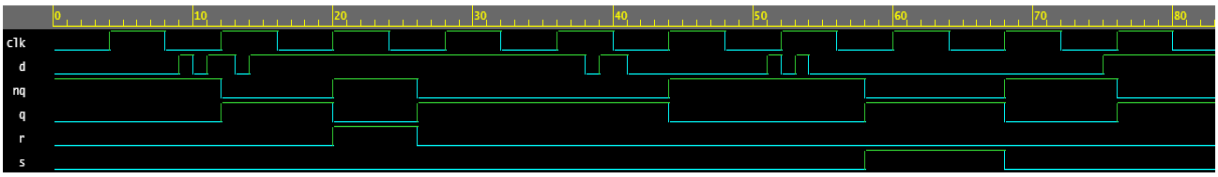


*Figura 12. Simulació amb EDA Playground d'un Flip-Flop D amb Set Reset*

Pel que fa al codi del Flip Flop es té en compte que sempre que el set o el reset passin de 0 lògic a 1 lògic el valor de la sortida Q es passi a ser 1 o 0 lògic respectivament. En el cas que aquestes dues variables tinguin un estat de 0 lògic el valor de Q es correspondrà amb el de l'entrada D quan li entri un senyal de rellotge.

Per controlar el correcte funcionament del flip-flop s'ha utilitzat una estructura If/Else, que permet canviar l'estat de la sortida Q en funció de l'estat de les altres variables.

## 4040

- **Entrades:** CLK, RST
- **Sortides:** a, b, c, d, e, f, g, h, i, j, k, l

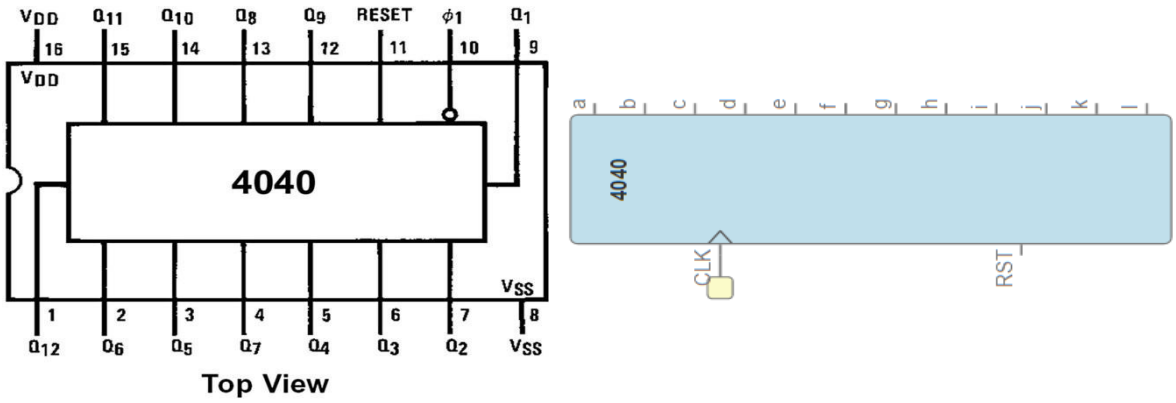


*Figura 13. Bloc 4040 comercial i dissenyat amb IceStudio*

El bloc 4040 és un comptador binari que va augmentant cada vegada que el rellotge (CLK) té un flanc positiu. L'entrada RST funciona com un reset i s'activa amb un flanc negatiu a l'entrada. Si l'estat del reset és baix el comptador es queda a zero.

Pel que fa a les sortides, "a" és el LSB i "l" el MSB, el comptador és de 12 bits i per tant pot comptar fins a $2^{12}$ - 1, que en decimal equival a 4095 unitats.

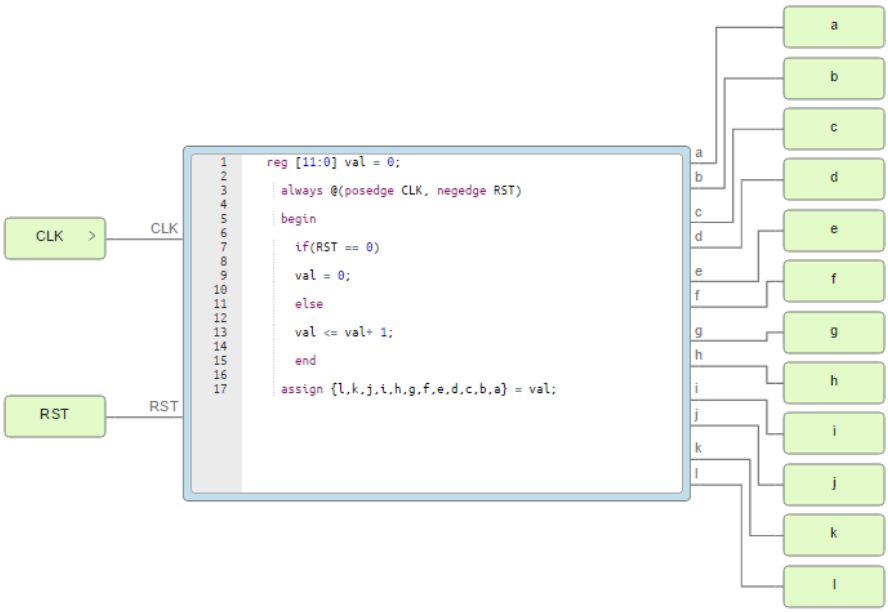


*Figura 14. Disseny del Comptador 4040 amb IceStudio*

Per comprovar el correcte funcionament del comptador es fa la simulació a EDA Playground i s'observa que en un període de 100 segons, amb 50 flancs positius de rellotge finalment queden en estat alt les sortides f, e i b que combinades sumen $2^5$ (f) + $2^4$ (e) + $2^1$ (b) = 32+16+2= 50. El reset a més sempre ha d'estar en estat alt per tal de permetre el correcte funcionament del comptador.

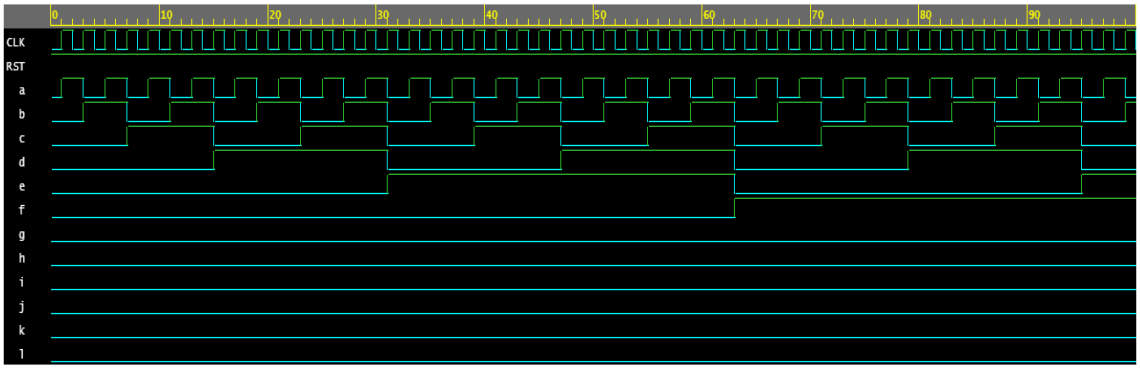


*Figura 15. Simulació amb EDA playground del comptador 4040*

Pel que fa al codi cal destacar que el reset es pot activar de forma asíncrona en qualsevol moment, ja que s'entrarà al bucle sempre que hi hagi un flanc positiu del rellotge o un flanc negatiu del reset.

Dins del bucle, mitjançant l'estructura condicional If/Else, s'augmenta el valor en una unitat cada vegada que hi ha un flanc de rellotge positiu o bé s'estableix que el valor és zero si el senyal de reset està en estat baix.

Finalment, s'assigna el valor corresponent a les sortides.

## 4071

- **Entrades:** A1, B1, A2, B2, A3, B3, A4, B4
- **Sortides:** Q1, Q2, Q3, Q4

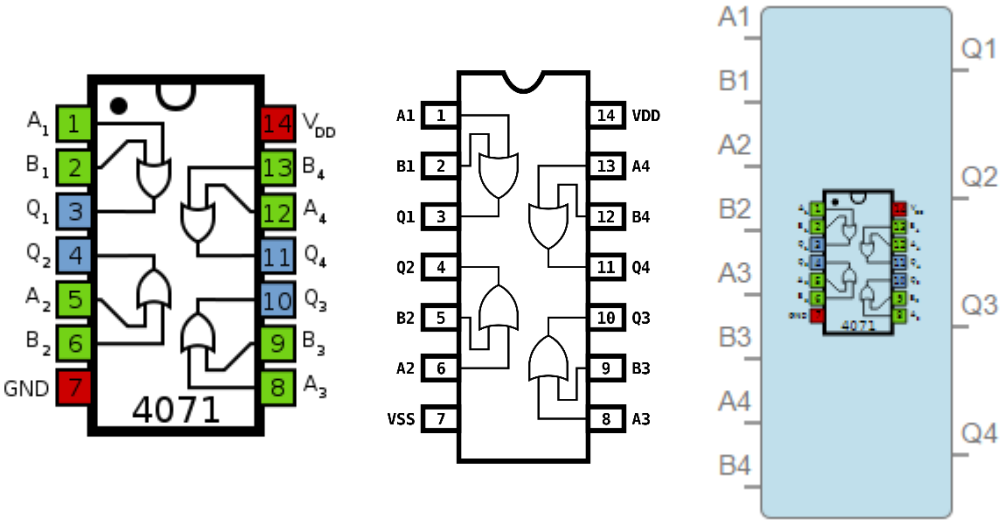


*Figura 16. Bloc 4071 comercial i dissenyat amb IceStudio*

El bloc 4071 és un conjunt de 4 portes OR de dues entrades, les entrades i sortides amb el mateix número pertanyen a la mateixa porta, així doncs, Q1 és la sortida d'una porta OR amb entrades A1 i B1. El disseny intern s'ha fet utilitzant les portes OR que té el programa.

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

*Taula 3. Taula de a veritat porta OR de dos entrades*

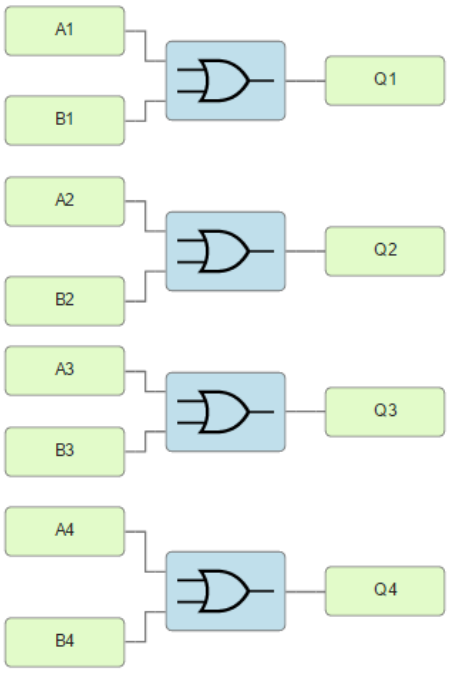


*Figura 17. Disseny intern bloc 4071*

## 4081

- **Entrades:** A1, B1, A2, B2, A3, B3, A4, B4
- **Sortides:** Q1, Q2, Q3, Q4

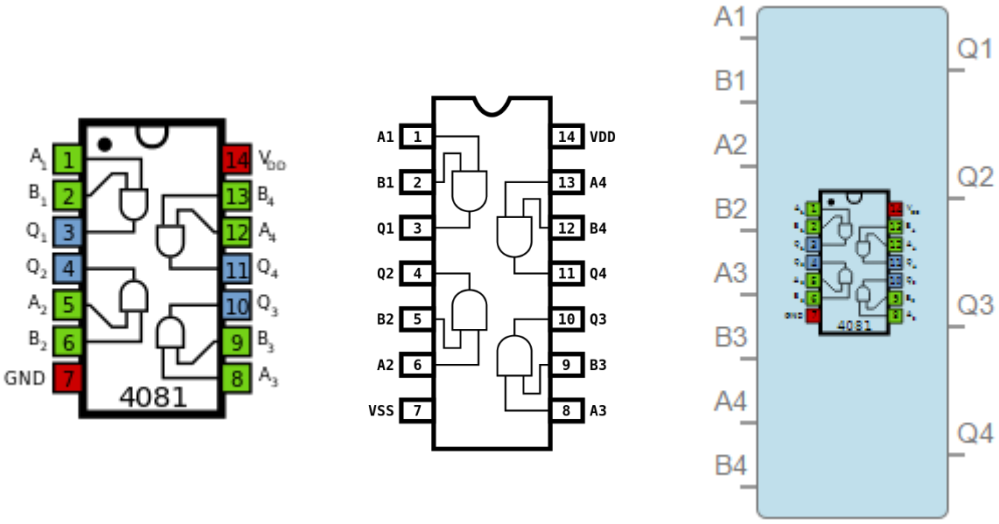


*Figura 18. Bloc 4081 comercial i dissenyat amb IceStudio*

El bloc 4081 és un conjunt de 4 portes AND de dues entrades en aquest cas les portes tenen les entrades A i B i la sortida Q. Cada porta té un número assignat d'1 a 4. Per tant, A1, B1 i Q1 són les entrades i la sortida d'una porta, A2, B2 i Q2 d'una altra i així amb les 4 portes.

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Taula 4. Taula de la veritat porta AND*

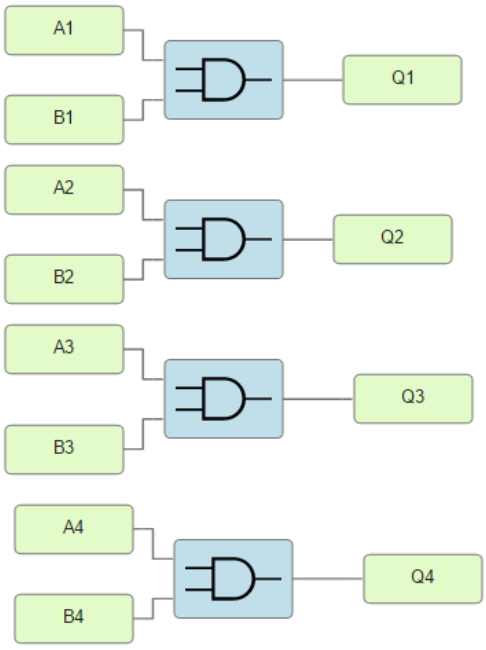


*Figura 19. Disseny intern bloc 4081*

## 4511

- **Entrades:** A, B, C, D, LT, BL, LE
- **Sortides:** a, b, c, d, e, f, g

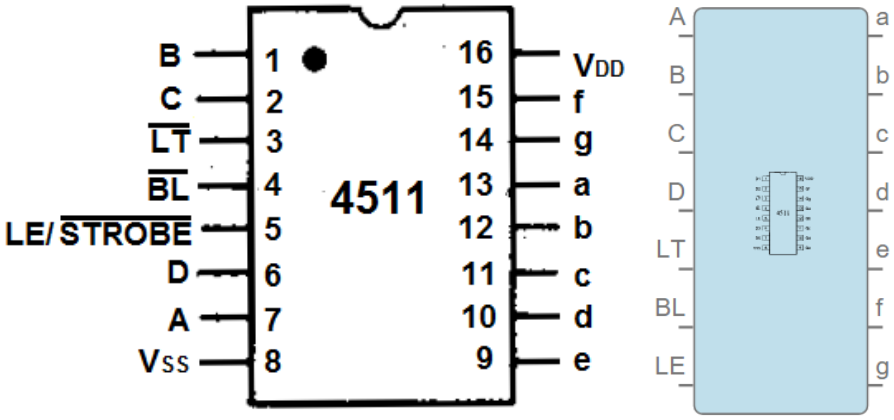


*Figura 20. Bloc 4511 comercial i bloc 4511 dissenyat amb IceStudio*

En aquest cas, les entrades es podrien dividir en dos grups, en primer lloc A, B, C i D que són els valors d'entrada del nombre binari on D és el MSB. LT, BE i LE són 3 variables de control sobre el funcionament del 4511. En cas que BL i LT siguin 1 lògic i LE 0 lògic el bloc convertirà el nombre binari (D,C,B,A) en una sortida de 7 segments, mentre que en la resta de casos les sortides estaran totes a 0 lògic o a 1 lògic depenent dels valors que tinguin BL i LT, però el convertidor BCD a 7 segments no funcionarà com a tal.

El CMOS 4511 és un circuit integrat que passa de BCD a 7 segments, i que per tant permet mostrar un nombre entre 0 i 9 en un display de 7 segments a partir d'una entrada binària.

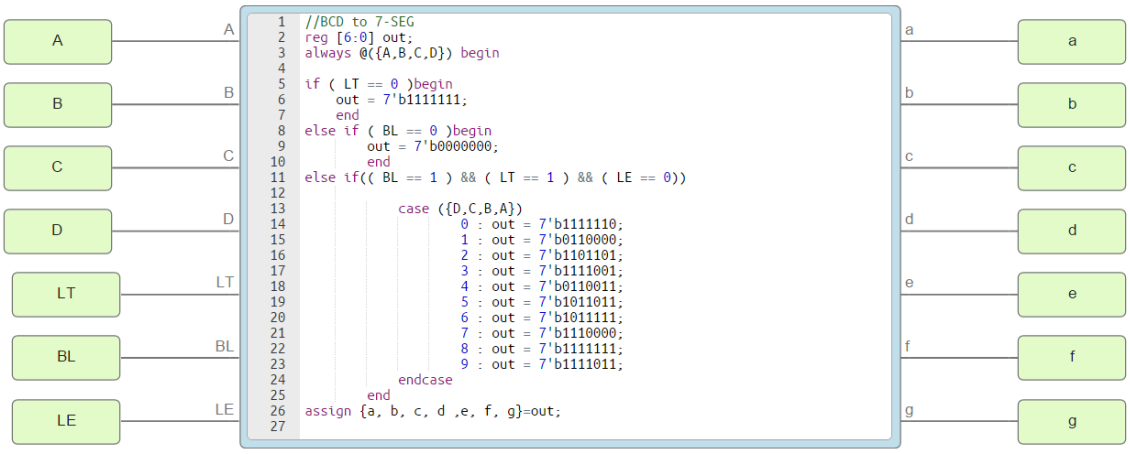


*Figura 21. Bloc BCD 7 Segments IceStudio*

Les sortides del bloc 4511 en aquest cas fan referència als 7 segments que es mostren en la figura X, i els nombres tindran la forma que es pot observar en la mateixa figura.

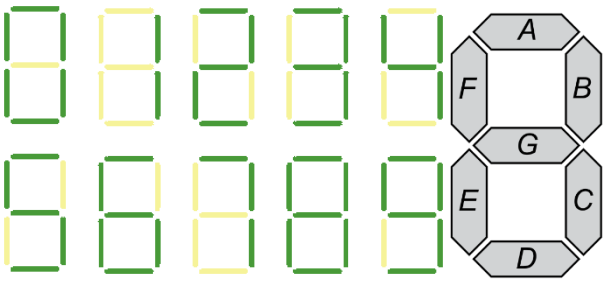


*Figura 22. Nomenclatura dels segments i forma dels nombres*

En general per fer servir el CMOS 4511 les variables de control tindran els valors que fan que el bloc passi d'un nombre binari a un display de 7 segments, el funcionament del bloc en aquest cas és el que s'observa en la simulació de la Figura 23.

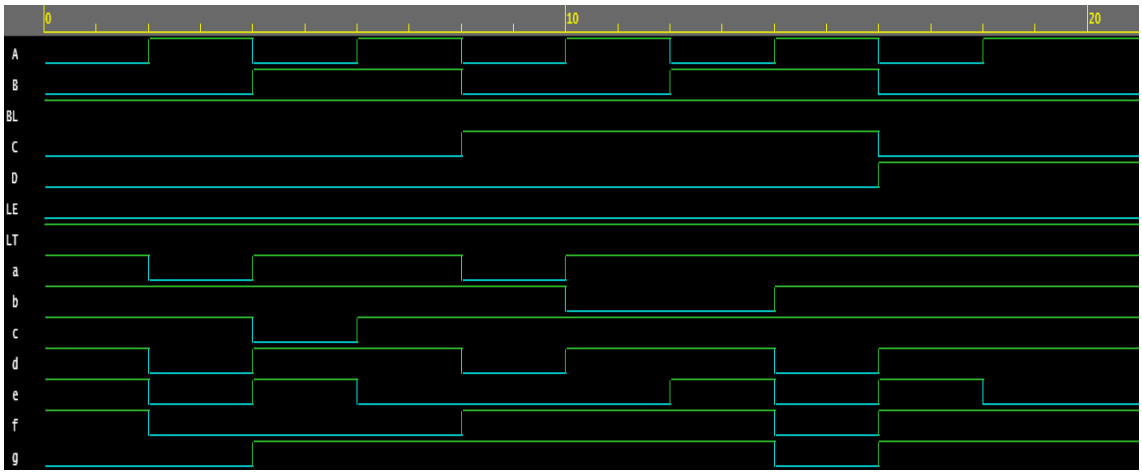


*Figura 23. Senyals d'entrada i de sortida del 4511 simulades amb EDA Playground*

El codi implementat funciona en 3 passos:

- Línia 1-10: Es crea una variable "out" que serà un nombre binari de 7 bits on es posarà 1 o 0 lògic depenent de l'estat del segment de sortida pertinent. A més d'això es comprova l'estat de les variables de control, en cas que siguin BL=1, LT=1 i LE 0 es va a la segona part del codi.

- Línia 11-25: S'assigna un valor a la variable "out" amb una estructura case que té com a variable el nombre binari d'entrada, i assigna a la variable "out" un valor binaria de 7 bits que realment reflecteix l'estat dels 7 segments, és a dir 0110000 voldria dir que els segments b i c s'il·luminen (número 1) i s'escriu el nombre binari per cada cas entre 0 i 9 d'acord amb el que s'ha mostrat a la figura 23.

- L'última part és assignar el valor de la variable "out" a les sortides del bloc, procés que es fa en l'última línia de codi, es fa sempre independentment del valor de les variables de control.

## Bloc de 4 displays

Per aprofitar al màxim les característiques de la placa d'entrenament, el darrer disseny és un bloc que fa que es puguin veure números als 4 displays de la part superior de forma simultània. Com a conseqüència del disseny de la placa només és possible mostar un dels 4 displays, mai 2 o més al mateix temps. Tot i que si es va canviant el display mostrat amb freqüències elevades l'ull humà té la impressió de veure'ls tots al mateix temps.

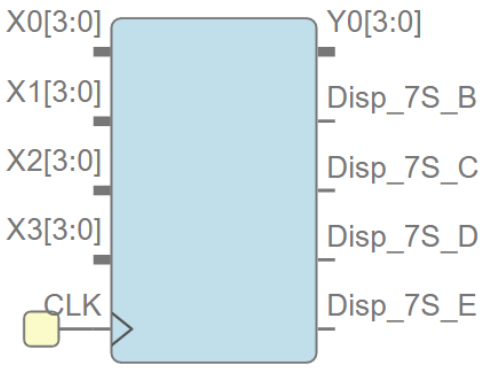


*Figura 24. Bloc 4 displays dissenyat amb IceStudio*

- **Entrades:** X0, X1, X2, X3, CLK
- **Sortides:** Y0, Disp_7S_B, Disp_7S_C, Disp_7S_D, Disp_7S_E.

Pel que fa a les entrades cal distingir entre el CLK i la resta, les entrades X1, X2, X3 i X4 són números binaris de 4 bits, i cadascun sortirà per un display diferent. CLK (rellotge) marca la freqüència amb que s'aniran permutant els displays, en general serà una freqüència de 150 Hz per simular que es veuen els 4 displays al mateix temps.

Pel que fa a les sortides Y0 sempre serà un nombre binari de 4 bits equivalent a X1, X2, X3 o X4. Les sortides de Disp indiquen el display on es veurà el número després de convertir-lo de binari a decimal.

Tot i no ser un bloc comercial, en aquest cas és molt interesant, ja que a més només fa falta utilitzar 8 entrades per fer servir 4 displays. Per exemple el display de la part inferior de la placa d'entrenament fa servir també 8 entrades, així doncs l'estalvi de pins és considerable.

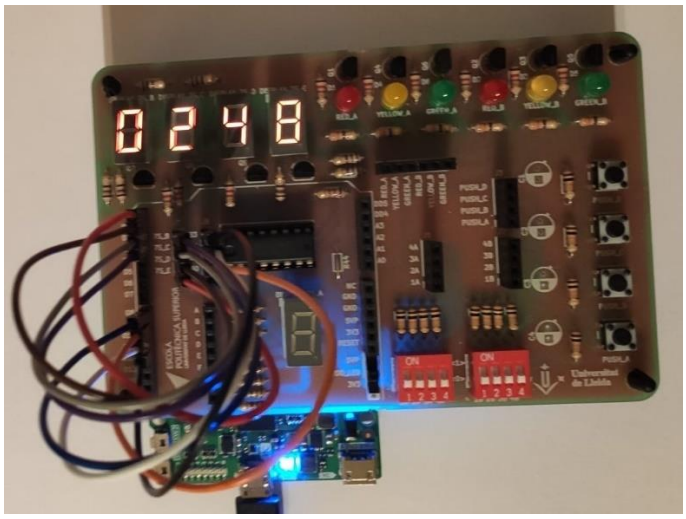


*Figura 25. Placa d'entrenamnet amb els 4 displays funcionant*

La Figura 25 mostra es connexions i els 4 displays funcionant aparentment tots al mateix temps degut a la elevada freqüència amb que varien els displays de sortida. Si s'agafa com a referència la Figura 25 els displays ordenats de dreta a esquerra serien 7S_B, 7S_C, 7S_D i 7S_E.

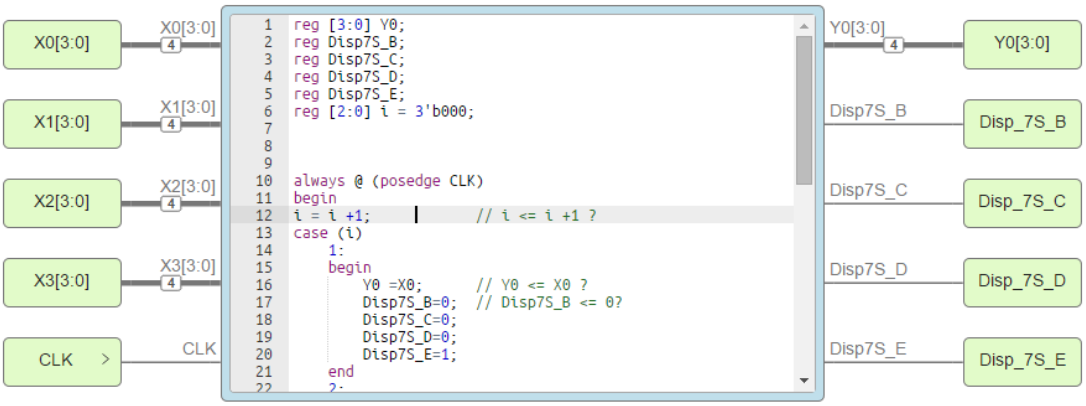


*Figura 26. Disseny intern bloc 4 displays*

El codi intern és relativament senzill. Quan CLK té un flanc positiu s'entra en un bucle que únicament conté una estructura case i un paràmetre de control que en aquest cas es la variable "i". La variable i fa que el case funcioni de forma cíclica (1-2-3-4-1-2-3-4...). Cada possible cas de l'estructura case té un display de sortida, a més la sortida Y0 en per cada valor del paràmetre "i" agafa un valor d'entrada X diferent. D'aquesta forma s'aconsegueix mostrar els 4 nombres diferents de forma cíclica. El codi complert es pot veure a l'annex.

# Implementació d'un rellotge amb alarma

## Introducció

La darrera part del treball és aconseguir un disseny funcional a patir d'algun dels blocs dissenyats. S'ha escollit fer un rellotge amb una alarma, ja que és una pràctica on s'aprofiten al màxim les característiques de la placa d'entrenament.

L'objectiu de la pràctica és aconseguir que els 4 displays de la placa d'entrenament mostrin els minuts i els segons. Per mostrar les hores s'utilitzaran els LEDs de la placa d'entrenament i per tant serà en codi binari.

Finalment amb els interruptors DIP serà possible programar una alarma que s'activarà encenent els 8 LEDs de la placa IceZUM Alhambra II.

Un cop establerts els objectius finals cal seguir un ordre en el disseny que serà el següent:

- Disseny de les unitats i desenes de segon
- Disseny de les unitats i desenes de minut i de les hores amb LEDs
- Disseny de l'alarma

Per la implementació d'un rellotge amb alarma s'han utilitzat només components dissenyats en aquest projecte i alguns elements que són accessibles dins del mateix programa o en llibreries creades per altres usuaris com en el cas dels rellotges i els busos de dades.

## Disseny de les unitats i desenes de segon

El disseny tant de les unitats com de les desenes de segons es pot veure a la Figura 27. Per tal de poder implementar-lo són necessaris dos comptadors 4040, dos busos agregadors de 4 bits, un bus separador de 4 bits, un bloc 4011 i dos rellotges.

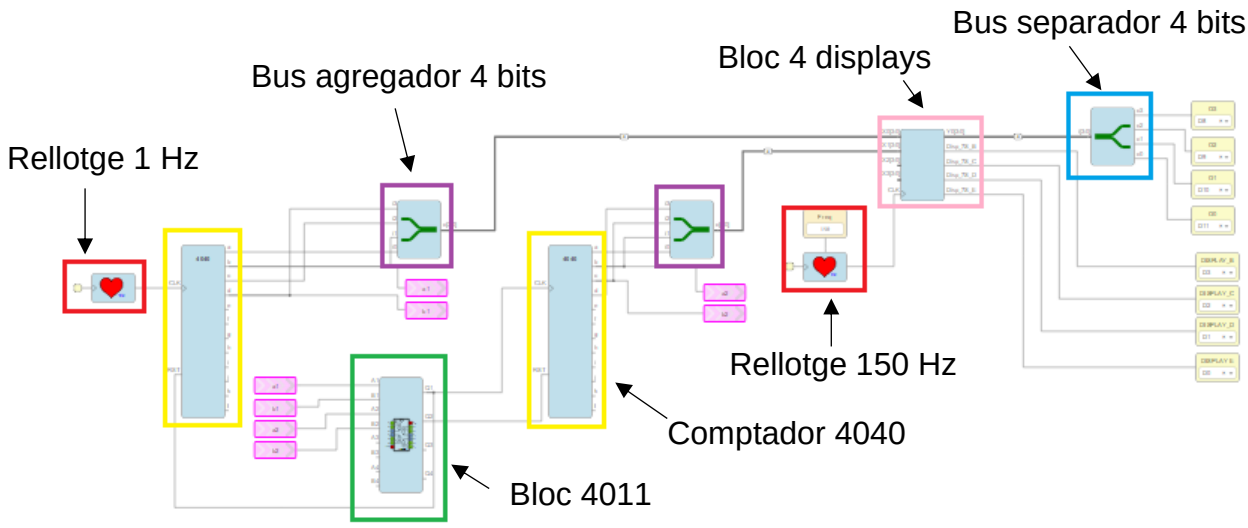


*Figura 27. Disseny dels segons del rellotge*

El primer pas per fer el rellotge funcional amb capacitat per comptar segons és posar al circuit un rellotge amb una freqüència d'un Hertz. Aquest primer rellotge és l'entrada al 4040 que compta les unitats dels segons. Situat a la part esquerra de la Figura 27.

Com els segons van de 0 a 9 de forma cíclica el reset ha de passar a estat baix cada vegada que el rellotge arribi a 10 (d'aquesta manera en lloc de 10 sortirà un 0 novament).

Per controlar que el reset passi a estat baix cada 10 segons i després torni a estat alt s'utilitza una porta NAND de dues entrades com les que conté el bloc 4011.

Les entrades de la porta NAND que controla el reset de les unitats dels segons són les sortides "b" i "d" del bloc 4040 de la part esquerra de la Figura 27. Les sortides "b" i "d" tenen un pes de $2^1$ i $2^3$ bits respectivament, és a dir 2 i 8 en sistema decimal. Per tant quan les dues sortides estan actives (cada 10 segons) el reset passa a estat baix i per tant el valor del comptador passa a ser 0.

Fins que el programa s'aturi el rellotge anirà comptant els segons, no existeix la possibilitat d'aturar-lo.

Un cop s'ha aconseguit comptar les unitats dels segons ara cal mostrar-les al lloc corresponent del display de 4 números de la placa d'entrenament. En aquest cas la distribució dels números és la següent:

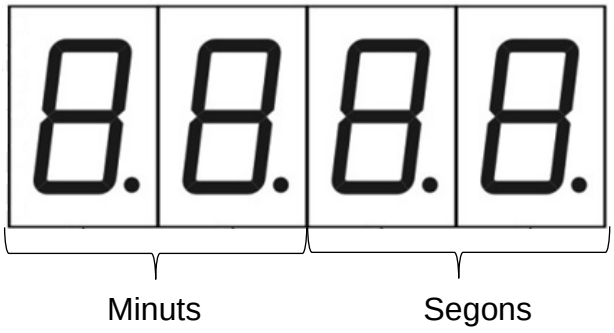


*Figura 28. Disposició elements rellotge*

Així doncs, amb les característiques que té el bloc de 4 displays, enquadrat de color rosa a la Figura 27, fa falta convertir les 4 primeres sortides del comptador a un bus d'un sol canal. Per fer-ho s'ha utilitzat un bus agregador de 4 bits que en el cas de les unitats dels segons es connecta a l'entrada superior del bloc de 4 displays.

Això és suficient per tenir les unitats de segon. Pel que fa a les desenes de segon el disseny de components és el mateix, únicament varien les entrades tant al rellotge com al reset del comptador 4040. En el cas del rellotge va connectat al reset del comptador d'unitats de segon. D'aquesta forma, cada vegada que les unitats de segon passin a zero el valor de les desenes de segon augmentarà en una unitat.

El reset en aquest cas anirà connectat a les sortides "b" i "c" del comptador 4040 situat més a la dreta, que tenen un valor de de $2^1$ i $2^2$ bits respectivament, és a dir 2 i 4 en binari. D'aquesta forma cada 60 segons es farà un reset a les desenes dels segons. S'aprofita el mateix bloc 4011, ja que disposa de 4 portes NAND i per tant és possible controlar fins a 4 resets amb un sol bloc. Els busos agragadors van a les entrades X0 (unitats de segon) i X1 (desenes de segon) del bloc de 4 displays per aconseguir que es situïn a la part dreta tal com es veu a la Figura 28 i d'acord amb el codi del bloc de 4 displays.

## Disseny de les unitats i desenes de minut i de les hores amb LEDs

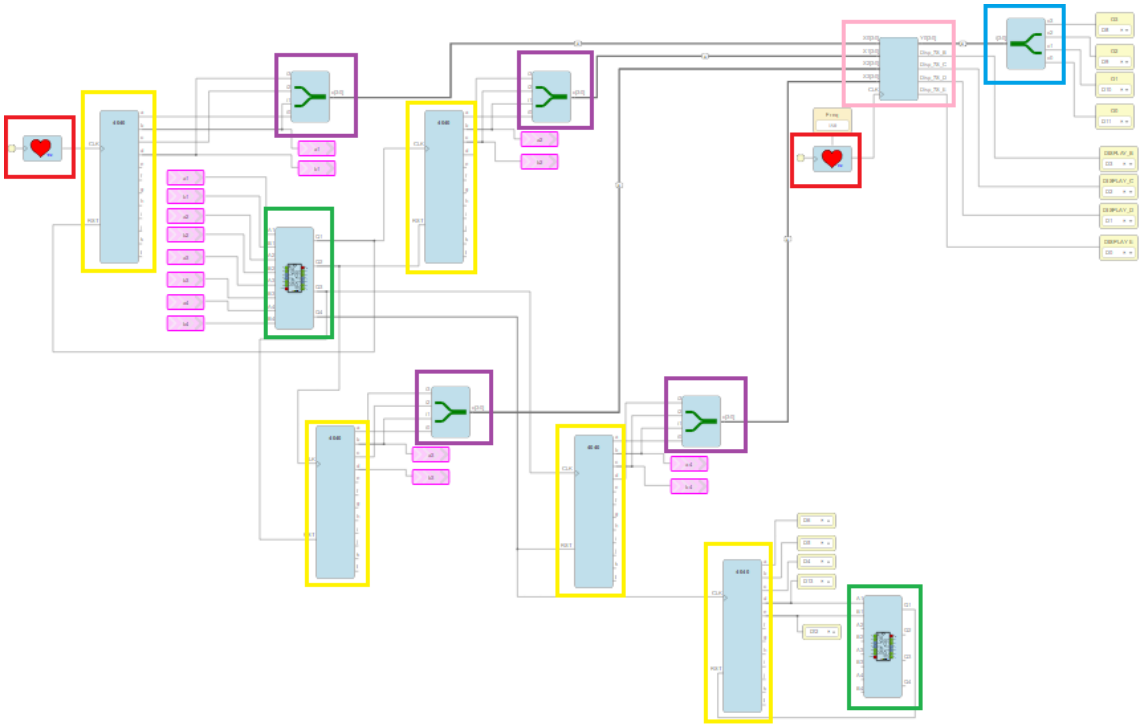

*Figura 29. Disseny del rellotge amb segons, minuts i hores.*

A la figura 29 es pot observar que no hi ha cap element diferent respecte a la Figura 28, si bé és cert que s'han afegit 3 comptadors, un altre bloc 4011 i dos busos agregadors més.

Pel que fa als minuts funcionen d'una forma semblant als segons, és a dir, utilitzen els 4 primers bits del comptador 4040 corresponent a les unitats i desenes de minuts. En el cas de les unitats de minut, situades en el tercer display començant per l'esquerra utilitzen com a entrada CLK l'entrada RST de les desenes de segons. Per tant cada 60 segons augmentaran en una unitat. De la mateixa forma que les unitats de segon, les unitats de minut tenen les sortides "b" i "d" connectades al bloc 4011 per generar un senyal de reset en aquest cas cada 10 minuts.

Seguint amb la mateixa dinàmica les desenes de minut tenen com a senyal d'entrada CLK el mateix senyal que serveix de reset a les unitats de minuts, per tant com era d'esperar augmentarà en una unitat cada 10 minuts. El seu reset es fa amb les sortides "b" i "c" del comptador de desenes de minuts i per tant es produirà cada 60 minuts.

Tant els segons com els minuts (unitats i desenes) fan servir busos agregadors per fer arribar el valor de sortida del comptador al bloc de 4 displays. Així doncs, els 4 displays serveixen en aquesta pràctica per representar els minuts i els segons.

Pel que fa a les hores, en canvi, cal utilitzar els LED, donant diferents pesos a cadascun d'ells per establir un criteri per comptar hores. En aquest cas s'utilitzaran els valors dels LEDs tal com es pot observar a la Figura 29.

| | | | | | |
|---|---|---|---|---|---|
| | $2^4$ = 16 h | $2^3$ = 8h | $2^2$ = 4h | $2^1$ = 2h | $2^0$ = 1h |

*Figura 30. Valor en hores dels LED en sistema decimal*

Si hi ha més d'un LED encès les hores es sumaran, si estan tots apagats és degut al fet que encara no s'ha complert una hora des que s'ha encès el rellotge. Pel que fa a la implementació és força similar a la dels minuts i els segons. En aquest cas el comptador encarregat de les hores rep com a entrada CLK el senyal de reset de les desenes de minuts, per tant cada 60 minuts augmentarà en una unitat. El senyal de reset es genera amb el bloc 4011 de la part inferior de la Figura 29, que té com a entrades les sortides del comptador 4040 "d" i "e", que en decimal equivalen a 24 si es sumen.

Així doncs, el reset es produirà cada 24 hores. Pel que fa a les sortides del comptador 4040 van directament a pins de sortida que s'han de connectar als LEDs. Així doncs una recomanació de connexions seria la següent:

| D0 | D1 | D2 | D3 | D8 | D9 | D10 | D11 |
|---|---|---|---|---|---|---|---|
| 7S_E | 7S_D | 7S_C | 7S_B | B0 | B1 | B2 | B3 |

*Taula 5. Connexions dels 4 displays per mostrar hores i minuts.*

| D4 | D5 | D6 | D12 | D13 |
|---|---|---|---|---|
| GREEN_B | YELLOW_B | RED_B | GREEN_A | YELLOW_A |

*Taula 6. Connexions per implementar les hores a la placa d'entrenament*

## Disseny de l'alarma

L'alarma està dissenyada per poder programar-se des dels interruptors DIP, i es podrà programar una hora determinada per activar-la (sense tenir en compte els minuts i segons).

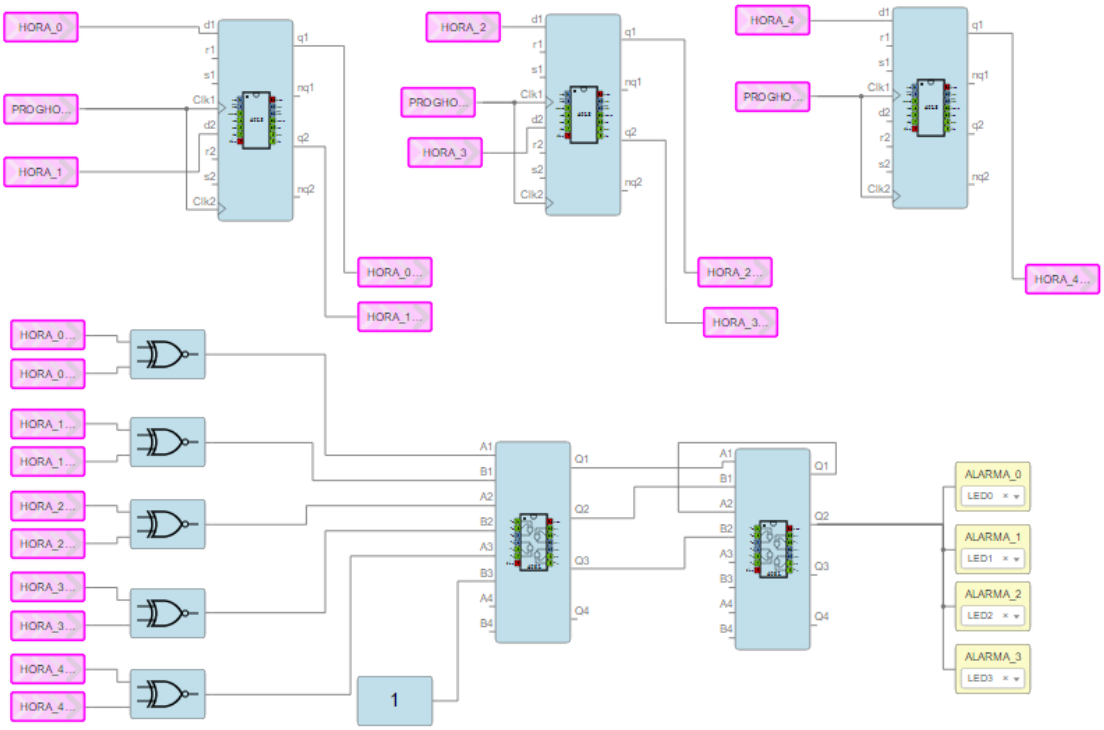


*Figura 31. Blocs utilitzats pel disseny de l'alarma*

Per fer el disseny de l'alarma hi ha dues parts molt diferenciades, la primera els blocs 4013 de la part superior que serveixen per fixar una hora perquè soni alarma. Per fer-ho cal activar l'entrada CLK, que anirà connectada a un polsador de forma que quan aquest sigui polsat l'alarma quedarà fixada a l'hora que indiquin els interruptors DIP, que tenen un funcionament pel que fa a pes de cada interruptor similar al dels LEDs per mostrar les hores, sent B4 el LSB i A4 el MSB, que tindran un pes d'1 h i 16 h respectivament.

A B

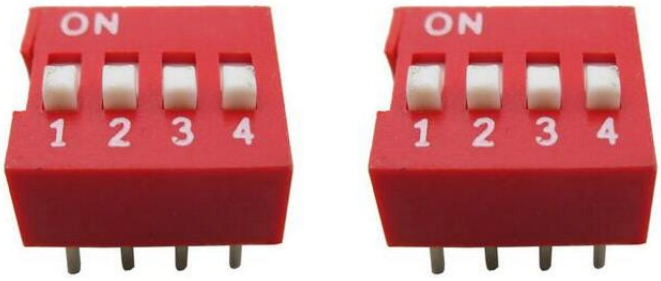


*Figura 32. Interruptors DIP de la placa d'entrenament*

La part inferior de la Figura 31 consta de dos blocs 4081 i un conjunt de portes XNOR que comproven si l'hora que s'ha programat a través dels flip-flops coincideix o no amb l'hora actual del rellotge, ja que les portes XNOR funcionen de la següent forma:

| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Taula 7. Taula de la veritat porta XNOR de dues entrades*

Així doncs, quan tots l'hora del relotge coincideixi amb l'hora d'alarma programada s'activarà l'alarma encenent 4 LEDs verds de la placa IceZUM Alhambra II. Com es pot observar a la Figura 33, que té una alarma programada a les 17 hores.

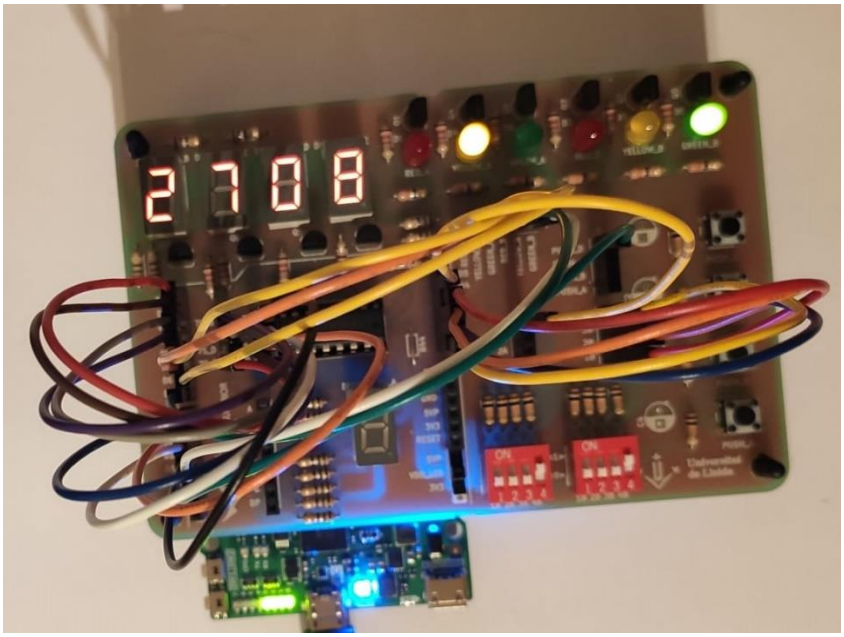


*Figura 33. Resultat final de la pràctica amb alarma programada a les 17 hores*

Les connexions recomanades serien les següents:

| DD0 (A0) | DD1 (A1) | DD2 (A2) | DD3 (A3) | DD4 |
|---|---|---|---|---|
| 4B | 3B | 2B | 1B | 4A |

## Conclusions

El primer que cal valorar a les conclusions és si s'han pogut assolir els objectius principals del treball. La resposta és sí, ja que ha estat possible dissenyar un nombre considerable de components de la família CMOS 4000 i, a més, utilitzant únicament software i hardware lliure.

La implementació d'un rellotge amb alarma demostra que amb pocs components i un disseny relativament simple es poden fer pràctiques que amb el model clàssic d'aprenentatge serien molt complexos d'implementar a un hardware. Així doncs, es pot demanar als estudiant que aprenguin a programar i implementar pràctiques d'una dificultat més elevada i també més interessants de traslladar a un hardware.

Cal també ser optimista amb les opcions d'evolucionar que té aquesta llibreria, ja que qualsevol usuari pot afegir els components que necessiti i per tant és un treball que pot seguir evolucionant a curt i mitjà termini.

Un altre aspecte important a analitzar és la viabilitat de fer pràctiques fora de les aules, amb les eines del programa IceStorm i una placa d'entrenament. En aquest aspecte el software i hardware lliure fan que sigui una opció més que raonable, ja que la universitat pot cedir les plaques als estudiants i aquests treballar des de casa, cosa que amb el model actual de pràctiques que es basava en el software privat Proteus era del tot impossible sense assumir l'elevat cost de la llicència.

Cal a més remarcar que el llenguatge Verilog, tot i ser desconegut per la majoria dels estudiants o usuaris que no estiguin habituats a treballar amb HDL, és relativament senzill d'entendre per la seva similitud amb C. Això pot semblar poc rellevant, però és un factor important a tenir en compte a l'hora de potenciar el desenvolupament de llibreries. Fins i tot es podria fer una pràctica consistent en implementar un bloc comercial. Realment possibilita diferents línies de treball interessants.

# REFERÈNCIES

[1] Field-programmable gate array [Online] Disponible:https://es.wikipedia.org/wiki/Field-programmable_gate_array [Accés: 2 de juliol de 2020]

[2] Parallel computing [Online] Disponible:https://en.wikipedia.org/wiki/Parallel_computing [Accés: 2 de juliol de 2020]

[3] Qué es un FPGA: Características y utilidades de este tipo de componentes [Online] Disponible:https://hardzone.es/reportajes/que-es/fpga-caracteristicas-utilidad/ [Accés: 2 de juliol de 2020]

[4] Soft microprocessor [Online] Disponible:https://ca.wikipedia.org/wiki/Soft_microprocessor [Accés 2 de juliol de 2020]

[5] Lenguaje de descripción de hardware [Online] Disponible:https://es.wikipedia.org/wiki/Lenguaje_de_descripci%C3%B3n_de_hardware [Accés 2 de juliol de 2020]

[6] Presentación VHDL [Online] Disponible:http://www1.frm.utn.edu.ar/tecnicad1/_private/Apuntes/VHDLPresentaci%C3%B3n2.0%5B1%5D.pdf [Accés: 2 de juliol de 2020]

[7] Principios Electrónicos: Lenguaje HDL [Online] Disponible: https://es.slideshare.net/Chichico_San/unidad-4-lenguaje-hdl-pe-isc [Accés 2 de juliol de 2020]

[8] VHDL [Online] Disponible: https://es.wikipedia.org/wiki/VHDL [Accés: 2 de juliol de 2020]

[9] Programación en VHDL [Online] Disponibe:https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_VHDL [Accés 2 de juliol de 2020]

[10] Introducción a HDL Verilog [Online] Disponible:https://www.dte.us.es/Members/paulino/Verilog-Intro.pdf [Accés: 2 de juliol de 2020]

[11] Verilog [Online] Disponible: https://en.wikipedia.org/wiki/Verilog [Accés 2 de juliol de 2020]

[12] Tutorial Verilog [Online] Disponible: http://www.iuma.ulpgc.es/users/ nunez /clases-FdC/verilog/Verilog%20Tutorial%20v1.pdf [Accés 2 de juliol de 2020]

[13] Programación en Verilog [Online] Disponible:https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Verilog [Accés 2 de juliol de 2020]

[14]¿Qué es el open source? [Online] Disponible:https://www.redhat.com/es/topics/open-source/what-is-open-source [Accés 2 de juliol de 2020]

[15] Hardware libre [Online] Disponible: https://www.ecured.cu/Hardware_libre [Accés 2 de juliol de 2020]

[16] El software propietario: Ejemplos y ventajas [Online] Disponible: https://fp.uoc.fje.edu/blog/el-software-propietario-ejemplos-y-ventajas/ [Accés: 2 de juliol de 2020]

[17] A. Saiz-Vela, P. Fontova, T. Pallejà, M. Tresanchez, J.A. Garriga, C. Roig, "Plataforma de desarrollo de bajo coste para la implementación de circuitos digitales en FPGA libres", *2020 XIV Tecnologías Aplicadas a la Enseñanza de la Electrónica (Technologies Applied to Electronics Teaching) (TAEE)*, Porto, 2020

[18] IceStudio Revolutionary Editor [Online] Disponible :https://icestudio.io/ [Accés: 2 de juliol de 2020]

[19] IceZUM Alhambra Board [Online] Disponible:https://github.com/FPGAwars/icezum/wiki [Accés 2 de juliol de 2020]

[20] FPGAwars IceStudio [Online] Disponible:https://github.com/FPGAwars/icestudio/ [Accés 2 de juliol de 2020]

[21] FPGAwars Alhambra II FPGA [Online]
Disponible:https://github.com/FPGAwars/Alhambra-II-FPGA
[Accés 2 de juliol de 2020]

[22] Pinout IceZUM Alhambra II [Online] Disponible:
https://raw.githubusercontent.com/FPGAwars/Alhambra-II-FPGA/master/doc/pinout/Alhambra%20II%20V1.0A%20- %20Pinout_v1.0_rev%202.svg
[Accés: 2 de juliol de 2020]

[23] EDA Plyaground help [Online]
Disponible: https://eda-playground.readthedocs.io/en/latest/intro.html
[Accés: 2 de juliol de 2020]

[24] Serie 4000 [Online] Disponible: https://es.wikipedia.org/wiki/Serie_4000
[Accés: 2 de juliol de 2020]

[25] Curso de electrónica digital [Online]
Disponible:https://www.uv.es/~marinjl/electro/digital2.html
[Accés: 2 de juliol de 2020]

# ANNEX

## Fitxer Verilog del Flip Flop amb Set i Reset

```
module Ffsr ( input d, input r, input s, input clk,
 output q, output nq);
 reg q;
 always @ (clk or s or r)
  begin
  if (s)
  q <= 1'b1;
  else if (r)
 q <= 1'b0;
 else
  q <= d;
 end
 assign nq=~q;
 endmodule
```

## Fitxer testbench Flip Flop amb Set i Reset

```
module Ffsr_tb;

      reg d = 0;
      reg s = 0;
      reg r = 0;
      reg clk = 0;

      wire q;
      wire nq;

 Ffsr test1 (d,r,s,clk,q,nq);
 initial begin
  // File were to store the simulation results
 $dumpfile("Ffsr.vcd");
```

```
$dumpvars;
        d=0;          // t= 0
        #9 d=1;   // t= 9 canviem el valor de d: 0 --> 1
        #1 d=0;   // t= 10  d: 1 --> 0
        #1 d=1;   // t= 11  d: 0 --> 1
        #2 d=0;   // t= 13  d: 1 --> 0
        #1 d=1;   // t= 14  d: 0 --> 1
        // Mantindrem d= 1 fins t = 38

        #6 r=1;   // t= 20 Activem senyal de reset
        #6 r=0;   // t= 26 Desactivem senyal de reset

        #12d=0;   // t= 38 d: 1 --> 0
        #1 d=1;   // t= 39 d: 0 --> 1
        #2 d=0;   // t= 41 d: 1 --> 0

        #10 d=1;  // t= 51 d: 0 --> 1
        #1 d=0;   // t= 52 d: 1 --> 0
        #1 d=1;   // t= 53 d: 0 --> 1
        #1 d=0;   // t= 54 d: 1 --> 0
        // Mantindrem d = 0 fins a t = 75
        #4 s=1;   // t= 58 Activem senyal de set
        #10 s=0;  // t= 68 desactivem senyal de set
        #7 d=1;   // t= 75 d: 0 --> 1
 #8 $finish;
end
 always
     begin
       #4 clk=!clk;
     end
endmodule
```

**Fitxer Verilog Comptador 4040**

```
module Comptador (input CLK, input RST, output a, output b, output c,
 output d, output e, output f, output g, output h, output i, output j,
 output k, output l);
    reg [11:0] val = 0;
    always @(posedge CLK, negedge RST)
    begin
     if(RST == 0)
          val = 0;
     else
          val <= val+ 1;
     end
     assign {l,k,j,i,h,g,f,e,d,c,b,a} = val;
endmodule
```

**Fitxer testbench Comptador 4040**

```
module Comptador_tb;
 // Input/Output
 reg CLK;
 reg RST;
 wire a;
 wire b;
 wire c;
 wire d;
 wire e;
 wire f;
 wire g;
 wire h;
 wire i;
 wire j;
 wire k;
 wire l;
 Comptador test1(CLK,RST,a,b,c,d,e,f,g,h,i,j,k,l);
 initial begin
 // File were to store the simulation results
 $dumpfile("Comptador.vcd");
 $dumpvars;
  CLK = 0;
 RST = 1;
  #100  $finish;
end
always
     begin
       #1 CLK=!CLK;
     end
endmodule
```

**Fitxer verilog CMOS 4511**

```
module BCD7Seg (
 input A,  input B,  input C,  input D,  input LT,  input BL,  input LE,
output a, output b, output c,  output d,  output e, output f,  output g);
 //BCD to 7-SEG
 reg [6:0] out;
 always @({A,B,C,D}) begin
 if ( LT == 0 )begin
    out = 7'b1111111;
    end
 else if ( BL == 0 )begin
       out = 7'b0000000;
       end
 else if(( BL == 1 ) && ( LT == 1 ) && ( LE == 0))

                        case ({D,C,B,A})
                                    0 : out = 7'b1111110;
                                    1 : out = 7'b0110000;
                                    2 : out = 7'b1101101;
                                    3 : out = 7'b1111001;
                                    4 : out = 7'b0110011;
                                    5 : out = 7'b1011011;
                                    6 : out = 7'b1011111;
                                    7 : out = 7'b1110000;
                                    8 : out = 7'b1111111;
                                    9 : out = 7'b1111011;
                        endcase
                end
 assign {a, b, c, d ,e, f, g}=out;
endmodule
```

**Fitxer testbench CMOS 4511**

```
module BCD7Seg_tb;
 // Input/Output
 reg A;
 reg B;
 reg C;
 reg D;
 reg LT;
 reg BL;
 reg LE;
 wire a;
 wire b;
 wire c;
 wire d;
 wire e;
 wire f;
 wire g;
BCD7Seg test1 (A,B,C,D,BL,LT,LE,a,b,c,d,e,f,g);
 initial begin
  // File were to store the simulation results
  $dumpfile("BCD7Seg.vcd");
 $dumpvars;
  // TODO: initialize the registers here
  // e.g. value = 1;
  // e.g. #2 value = 0;
LT = 1;
BL = 1;
LE = 0;
A = 0;
B = 0;
C = 0;
```

```
D = 0;
//0
A = 0;
//1
#2 A=1;   // t= 2 canviem el valor de A: 0 --> 1
#2 A=0;   // t= 4  A: 1 --> 0
//2
 B=1;
//3
#2 A=1;
//4
#2 B=0;
#0 A=0;
#0 C = 1;
//5
#2 A=1;
#2 A=0;
//6
#2 B=1;
//7
#2 A=1;
//8
#2 B=0;
#0 A=0;
#0 C=0;
#0 D=1;
//9
#2 A=1;
 #8 $finish;
 end
endmodule
```

**Codi intern bloc dels 4 displays**

```
reg [3:0] Y0;
reg Disp7S_B;
reg Disp7S_C;
reg Disp7S_D;
reg Disp7S_E;
reg [2:0] i = 3'b000;

always @ (posedge CLK)
begin
i = i +1;          // i <= i +1 ?
case (i)
  1:
  begin
    Y0 =X0;     // Y0 <= X0 ?
    Disp7S_B=0;  // Disp7S_B <= 0?
    Disp7S_C=0;
    Disp7S_D=0;
    Disp7S_E=1;
  end
  2:
  begin
    Y0 =X1;
    Disp7S_B=0;
    Disp7S_C=0;
    Disp7S_D=1;
    Disp7S_E=0;
    end
```

```
    3:
    begin
      Y0 = X2;
      Disp7S_B=0;
      Disp7S_C=1;
      Disp7S_D=0;
      Disp7S_E=0;
    end
    4:
    begin
      Y0 = X3;
      Disp7S_B=1;
      Disp7S_C=0;
      Disp7S_D=0;
      Disp7S_E=0;
      i = 3'b000;  // i <= 3'b000
    end
    endcase
  end
```